

About This Book

This book is an addendum to the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition). All other 32-bit functions currently supported by OS/2 Warp (PowerPC Edition), are described in the Toolkit programming references.

There are no undocumented functions or methods such as those in the OEMI documentation. All interfaces will be documented in this publication. All OS/2 Warp Version 3 OEMI documentation has been added to this document or replaced with equivalent functions more appropriate to the IBM Microkernel-based architecture.

Who Should Read This Book

This book is intended for programmers developing OS/2 applications for OS/2 Warp (PowerPC Edition). It is not intended for developers of Shared Services for the IBM Microkernel-based architecture.

How This Book Is Organized

This book contains the following chapters:

[Keyboard](#) describes all of the Keyboard functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[Mouse](#) describes all of the Mouse functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[Video](#) describes all of the Video functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[Presentation Manager](#) describes the Presentation Manager functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition) that are new or enhanced on OS/2 Warp (PowerPC Edition).

[Input Method](#) describes all of the Input Method functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[Control Program](#) describes the Control Program functions and data types that are new or enhanced on OS/2 Warp (PowerPC Edition).

[RAM Semaphore Support](#) describes all of the RAM Semaphore functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[Network](#) describes all of the Network functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[Problem Determination](#) describes all of the Reliability, Availability and Serviceability (RAS) functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[Performance Services](#) describes all of the Performance Services functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

[OpenGL 3D Rendering](#) describes all of the OS/2-specific OpenGL functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

Introduction

OS/2 Warp (PowerPC Edition) supports all of the current 32-bit functions of OS/2 Warp Version 3. It does not support any 16-bit functions. In addition, OS/2 Warp (PowerPC Edition) provides the following *new* 32-bit functions:

- VIO/KBD/MOU text mode functions
- Input Method functions
- RAM Semaphore functions
- Network-independent functions
- RAS log and trace functions
- Performance services functions
- OpenGL OS/2-specific functions
- Unicode support functions
- Unicode wrappers for OS/2 functions

These functions are documented in each of these two books:

- *OS/2 Warp (PowerPC Edition) API Addendum Volume I*
 - VIO/KBD/MOU text mode functions
 - Input Method functions
 - RAM Semaphore functions
 - Network-independent functions
 - RAS log and trace functions
 - Performance services functions
 - OpenGL OS/2-specific functions
 - Enhancements to existing functions
- *OS/2 Warp (PowerPC Edition) API Addendum Volume II - Unicode Functions*
 - Unicode support functions
 - Unicode wrappers for OS/2 functions

OS/2 Warp (PowerPC Edition) does not support:

- 16-bit functions
- 16-bit (segmented) features that are variants of 32-bit functions
- Functions related to BIOS (not applicable to the PowerPC architecture)
- DosAllocThreadLocalMemory and DosFreeThreadLocalMemory (replaced by C run-time support)

Interfaces distributed previously that are not listed here are not supported by OS/2 Warp (PowerPC Edition) because either they are:

- 16-bit interfaces that have no meaning on a pure 32-bit base
- Obsolete

OS/2 Warp (PowerPC Edition) will provide a dynamically linked C-language run-time library that supports multithreaded and Unicode-enabled programs. For information on this library, see *C Library Reference* .

Keyboard

This chapter describes all of the 32-bit Keyboard functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

The chapter includes the following sections:

- [Keyboard Functions](#)
- [Keyboard Data Types](#)

Keyboard Functions

The keyboard functions described in this section are:

- [KbdCharIn](#)
- [KbdFlushBuffer](#)
- [KbdGetConsole](#)

- [KbdGetCp](#)
- [KbdGetHWID](#)
- [KbdGetLayout](#)
- [KbdGetLayoutUni](#)
- [KbdGetStatus](#)
- [KbdPeek](#)
- [KbdSetCp](#)
- [KbdSetLayout](#)
- [KbdSetLayoutUni](#)
- [KbdSetStatus](#)
- [KbdStringIn](#)
- [KbdXlate](#)

KbdCharIn

KbdCharIn - Syntax

KbdCharIn reads a character data record from the keyboard.

```
#define INCL_KBD
#include <os2.h>

PKBDKEYINFO    CharData; /* Pointer to character data. */
ULONG          Wait;     /* Wait Flag. */
HKBD           hkbd;     /* Reserved. Must be 0. */
APIRET         rc;       /* Return code. */

rc = KbdCharIn(CharData, Wait, hkbd);
```

KbdCharIn Parameter - CharData

CharData ([PKBDKEYINFO](#)) - output
Pointer to character data.

A pointer to a [KBDKEYINFO](#) structure in which the character data is returned.

KbdCharIn Parameter - Wait

Wait (ULONG) - input
Wait Flag.

0	IO_WAIT Wait for a keystroke if one is not available. The keystroke returned is removed from the queue.
1	IO_NOWAIT Return immediately, with or without a keystroke. If a keystroke is returned, remove it from the queue.

2	IO_PEEK Return immediately, with or without a keystroke. Do not remove the keystroke from the queue.
3	IO_PEEKWAIT Wait for a keystroke if one is not available. Return the keystroke but do not remove it from the queue.

KbdCharIn Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdCharIn Return Value - rc

rc (APIRET) - returns
Return code.

KbdCharIn returns one of the following values:

0	NO_ERROR
375	ERROR_KBD_INVALID_IOWAIT
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdCharIn - Parameters

CharData (PKBDKEYINFO) - output
Pointer to character data.

A pointer to a [KBDKEYINFO](#) structure in which the character data is returned.

Wait (ULONG) - input
Wait Flag.

0	IO_WAIT Wait for a keystroke if one is not available. The keystroke returned is removed from the queue.
1	IO_NOWAIT Return immediately, with or without a keystroke. If a keystroke is returned, remove it from the queue.
2	IO_PEEK Return immediately, with or without a keystroke. Do not remove the keystroke from the queue.
3	IO_PEEKWAIT Wait for a keystroke if one is not available. Return the keystroke but do not remove it from the queue.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdCharIn returns one of the following values:

0	NO_ERROR
375	ERROR_KBD_INVALID_IOWAIT

445
447
504

ERROR_KBD_FOCUS_REQUIRED
ERROR_KBD_KEYBOARD_BUSY
ERROR_KBD_EXTENDED_SG

KbdCharIn - Remarks

Note: KbdCharIn returns a complete keystroke. This behavior is unlike the OS/2 1.3 version, which returned only a single byte. This affects only double-byte character set (DBCS) characters.

If bit 0 of *fbStatus* is set, the character returned is either 0 or 0xe0. The Unicode character contains the virtual key.

For valid characters, the character in the current code page is returned, and the Unicode character contains the Unicode encoding of the character.

On an enhanced keyboard, the secondary enter key returns the normal character 0DH and a scan code of E0H.

Extended ASCII codes are identified by bit 1 of the status byte being set to on, and the ASCII character code being either 00H or E0H. Both conditions must be satisfied for the character to be an extended keystroke. For extended ASCII codes, the scan-code byte returned is the second code (extended code). Usually, the extended ASCII code is the scan code of the primary key that was pressed.

A thread in the foreground session that repeatedly polls the keyboard with KbdCharIn (with no wait), can prevent all regular priority-class threads from executing. If polling must be used, and a minimal amount of other processing is being performed, the thread should periodically yield to the CPU by issuing a DosSleep call for an interval of at least 5 milliseconds.

KbdCharIn - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

KbdFlushBuffer

KbdFlushBuffer - Syntax

KbdFlushBuffer removes all entries from the keyboard buffer. This discards all user keystrokes typed ahead.

```
#define INCL_KBD
#include <os2.h>

HKBD      hkbd; /* Reserved. Must be 0. */
APIRET     return; /* Return code. */

return = KbdFlushBuffer(hkbd);
```

KbdFlushBuffer Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdFlushBuffer Return Value - return

return (APIRET) - returns
Return code.

KbdFlushBuffer returns one of the following values:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdFlushBuffer - Parameters

hkbd (HKBD) - input
Reserved. Must be 0.

return (APIRET) - returns
Return code.

KbdFlushBuffer returns one of the following values:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdFlushBuffer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

KbdGetConsole

KbdGetConsole - Syntax

KbdGetConsole reads a key, mouse event, or a notification from the console.

```
#define INCL_KBD
#include <os2.h>

PVOID      Data;      /* Pointer to event data. */
PULONG     Kind;      /* Kind of event returned. */
ULONG      Flag;      /* Wait for a keystroke flag. */
HKBD       hkbd;      /* Reserved. Must be 0. */
APIRET     return;    /* Return code. */

return = KbdGetConsole(Data, Kind, Flag, hkbd);
```

KbdGetConsole Parameter - Data

Data (PVOID) - output
Pointer to event data.

A pointer to a location where the event data is returned. The data type returned depends on the kind of event returned. For keyboard events, the data type returned is [KBDKEYINFO](#). For mouse events, the data type returned is [MOUQUEINFO](#).

KbdGetConsole Parameter - Kind

Kind (PULONG) - output
Kind of event returned.

One of the following values is returned:

0	No event available
1	Keyboard event (not a valid character)
2	Character
3	Mouse event
4	Notification

KbdGetConsole Parameter - Flag

Flag (ULONG) - input
Wait for a keystroke flag.

The possible values are:

0	IO_WAIT Wait for an event if one is not available. The event returned is removed from queue.
---	---

1	IO_NOWAIT
	Return immediately, with or without an event. If an event is returned, removed it from the queue.
2	IO_PEEK
	Return immediately, with or without an event. Do not remove the event from the queue.
3	IO_PEEKWAIT
	Wait for an event if one is not available. Return the event but do not remove it from the queue.

KbdGetConsole Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdGetConsole Return Value - return

return (APIRET) - returns
Return code.

KbdGetConsole returns one of the following values:

0	NO_ERROR.
375	ERROR_KBD_INVALID_HANDLE
439	ERROR_KBD_INVALID_IOWAIT
464	ERROR_KBD_DETACHED

KbdGetConsole - Parameters

Data (PVOID) - output
Pointer to event data.

A pointer to a location where the event data is returned. The data type returned depends on the kind of event returned. For keyboard events, the data type returned is [KBDKEYINFO](#). For mouse events, the data type returned is [MOUQUEINFO](#).

Kind (PULONG) - output
Kind of event returned.

One of the following values is returned:

0	No event available
1	Keyboard event (not a valid character)
2	Character
3	Mouse event
4	Notification

Flag (ULONG) - input
Wait for a keystroke flag.

The possible values are:

0	IO_WAIT
	Wait for an event if one is not available. The event returned is removed from queue.
1	IO_NOWAIT
	Return immediately, with or without an event. If an event is returned, removed it from the queue.
2	IO_PEEK
	Return immediately, with or without an event. Do not remove the event from the queue.
3	IO_PEEKWAIT

Wait for an event if one is not available. Return the event but do not remove it from the queue.

hkbd (HKBD) - input
Reserved. Must be 0.

return (APIRET) - returns
Return code.

KbdGetConsole returns one of the following values:

0	NO_ERROR.
375	ERROR_KBD_INVALID_HANDLE
439	ERROR_KBD_INVALID_IOWAIT
464	ERROR_KBD_DETACHED

KbdGetConsole - Remarks

KbdGetConsole allows the retrieval of either a keyboard event or mouse event. This should be used by programs using both input devices. See [KbdCharIn](#) and [MouReadEventQue](#) for details of the returned values and conditions.

Only those mouse events enabled by the event mask are returned. By default, the event mask is disabled.

KbdGetConsole - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

KbdGetCp

KbdGetCp - Syntax

KbdGetCp gets the current keyboard code page.

```
#define INCL_KBD
#include <os2.h>

ULONG      ulReserved; /* Reserved. Must be 0. */
PUSHORT    pidCP;      /* Pointer to Code-page ID. */
HKBD       hkbd;       /* Reserved. Must be 0. */
APIRET     return;     /* Return code. */

return = KbdGetCp(ulReserved, pidCP, hkbd);
```

KbdGetCp Parameter - ulReserved

ulReserved (ULONG) - input
Reserved. Must be 0.

KbdGetCp Parameter - pidCP

pidCP (USHORT) - output
Pointer to Code-page ID.

A pointer to a USHORT in which the code-page ID is returned.

KbdGetCp Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdGetCp Return Value - return

return (APIRET) - returns
Return code.

KbdGetCp returns one of the following values:

0	NO_ERROR.
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdGetCp - Parameters

ulReserved (ULONG) - input
Reserved. Must be 0.

pidCP (USHORT) - output
Pointer to Code-page ID.

A pointer to a USHORT in which the code-page ID is returned.

hkbd (HKBD) - input
Reserved. Must be 0.

return (APIRET) - returns
Return code.

KbdGetCp returns one of the following values:

0	NO_ERROR.
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdGetCp - Remarks

The code-page ID is the currently active keyboard code page. A value of 0 indicates the default code-page translation table.

KbdGetCp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

KbdGetHWID

KbdGetHWID - Syntax

KbdGetHWID returns the type of keyboard in use. The hardware ID specifies the type of keyboard attached and is defined by the manufacturer.

```
#define INCL_KBD
#include <os2.h>

PKBDHWID    pkbdhwid; /* Pointer to hardware ID. */
HKBD        hkbd;     /* Reserved. Must be 0. */
APIRET      return;   /* Return code. */

return = KbdGetHWID(pkbdhwid, hkbd);
```

KbdGetHWID Parameter - pkbdhwid

pkbdhwid (**PKBDHWID**) - output
Pointer to hardware ID.

A pointer to a **KBDHWID** structure in which the keyboard hardware ID is returned.

KbdGetHWID Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdGetHWID Return Value - return

return (APIRET) - returns
Return code.

KbdGetHWID returns one of the following values:

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdGetHWID - Parameters

pkbdhwid (**PKBDHWID**) - output
Pointer to hardware ID.

A pointer to a **KBDHWID** structure in which the keyboard hardware ID is returned.

hkbd (HKBD) - input
Reserved. Must be 0.

return (APIRET) - returns
Return code.

KbdGetHWID returns one of the following values:

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdGetHWID - Remarks

The hardware ID indicates the basic layout of the keyboard, such as whether or not a numeric keypad exists. It gives no information about key values.

KbdGetHWID - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

KbdGetLayout

KbdGetLayout - Syntax

KbdGetLayout returns the name of the keyboard layout in use.

```
#include <os2.h>

PSZ      pszName; /* Keyboard layout name. */
HKBD     hkbd;    /* Reserved. Must be 0. */
APIRET   rc;      /* Return code. */

rc = KbdGetLayout(pszName, hkbd);
```

KbdGetLayout Parameter - pszName

pszName (PSZ) - output
Keyboard layout name.

A pointer to the location to return the keyboard layout name. This must be at least 9 bytes long.

KbdGetLayout Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdGetLayout Return Value - rc

rc (APIRET) - returns
Return code.

KbdGetHWID returns one of the following values:

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdGetLayout - Parameters

pszName (PSZ) - output
Keyboard layout name.

A pointer to the location to return the keyboard layout name. This must be at least 9 bytes long.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdGetHWID returns one of the following values:

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdGetLayout - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

KbdGetLayoutUni

KbdGetLayoutUni - Syntax

KbdGetLayout returns the name of the keyboard layout in use.

```
#include <os2.h>
```

```
USHORT    *name; /* Keyboard layout name. */  
HKBD      hkbd; /* Reserved. Must be 0. */  
APIRET    rc;    /* Return code. */
```

```
rc = KbdGetLayoutUni(name, hkbd);
```

KbdGetLayoutUni Parameter - name

name (USHORT *) - output
Keyboard layout name.

A pointer to the location to return the keyboard layout name in unicode. This must be at least 9 UniChars long.

KbdGetLayoutUni Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdGetLayoutUni Return Value - rc

rc (APIRET) - returns
Return code.

KbdGetHWID returns one of the following values:

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdGetLayoutUni - Parameters

name (USHORT *) - output
Keyboard layout name.

A pointer to the location to return the keyboard layout name in unicode. This must be at least 9 UniChars long.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdGetHWID returns one of the following values:

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdGetLayoutUni - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

KbdGetStatus

KbdGetStatus - Syntax

KbdGetStatus returns information about the keyboard.

```
#define INCL_KBD
#include <os2.h>

PKBDINFO    pkbinfo; /* Pointer to keyboard data. */
HKBD        hkbd;    /* Reserved. Must be 0. */
APIRET      rc;      /* Return code. */

rc = KbdGetStatus(pkbinfo, hkbd);
```

KbdGetStatus Parameter - pkbinfo

pkbinfo ([PKBDINFO](#)) - output
Pointer to keyboard data.

A pointer to a [KBDKEYINFO](#) structure in which the keyboard status is returned.

KbdGetStatus Parameter - hkbd

hkbd ([HKBD](#)) - input
Reserved. Must be 0.

KbdGetStatus Return Value - rc

rc (APIRET) - returns
Return code.

KbdGetStatus returns one of the following values:

0	NO_ERROR.
373	ERROR_KBD_PARAMETER
376	ERROR_KBD_INVALID_LENGTH
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdGetStatus - Parameters

pkbinfo ([PKBDINFO](#)) - output
Pointer to keyboard data.

A pointer to a [KBDKEYINFO](#) structure in which the keyboard status is returned.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdGetStatus returns one of the following values:

0	NO_ERROR.
373	ERROR_KBD_PARAMETER
376	ERROR_KBD_INVALID_LENGTH
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdGetStatus - Remarks

Some of the keyboard status information can be changed using [KbdSetStatus](#).

In [KBDINFO](#), the upper byte of *fsInterim* is the NLS shift state. The meaning of the NLS shift varies by language. The following bits are defined to access this data:

NLSS_NLS1	(0x01) - Full-width, National layer
NLSS_NLS2	(0x02) - Katakana, JAMO phonetic
NLSS_NLS3	(0x04) - Hiragana, Hangeul, TsangJye
NLSS_APPL	(0x10) - Application bit
NLSS_NLS4	(0x40) - Romanji, HanjaCsr
NLSS_KANJI	(0x80) - Kanji, Hanji

KbdGetStatus - Topics

Select an item:

KbdPeek

KbdPeek - Syntax

KbdPeek returns a keyboard-character data record, if available, but does not remove it from the queue.

```
#define INCL_KBD
#include <os2.h>

PKBDKEYINFO CharData; /* Pointer to character data. */
HKBD        hkbd;     /* Reserved. Must be 0. */
APIRET      rc;        /* Return code. */

rc = KbdPeek(CharData, hkbd);
```

KbdPeek Parameter - CharData

CharData ([PKBDKEYINFO](#)) - output
Pointer to character data.

A pointer to a [KBDKEYINFO](#) structure in which the character data is returned.

KbdPeek Parameter - hkbd

hkbd ([HKBD](#)) - input
Reserved. Must be 0.

KbdPeek Return Value - rc

rc ([APIRET](#)) - returns
Return code.

KbdPeek returns one of the following values:

0	NO_ERROR.
375	ERROR_KDB_INVALID_IOWAIT
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdPeek - Parameters

CharData ([PKBDKEYINFO](#)) - output
Pointer to character data.

A pointer to a [KBDKEYINFO](#) structure in which the character data is returned.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdPeek returns one of the following values:

0	NO_ERROR.
375	ERROR_KDB_INVALID_IOWAIT
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdPeek - Remarks

Note: KbdPeek returns a complete keystroke. This behavior is unlike the OS/2 1.3 version, which returned only a single byte. This is significant only for DBCS characters.

If bit 0 of *fbStatus* is set, the character returned is either 0 or 0xe0. The Unicode character contains the virtual key.

For valid characters, the character in the current code page is returned, and the Unicode character contains the Unicode encoding of the character.

KbdPeek - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

KbdSetCp

KbdSetCp - Syntax

KbdSetCp sets the current keyboard code page used to translate keystrokes received from the keyboard. This causes a change in the translation of keys. The code page can be any code page (except EBCDIC to DBCS). Setting this code page does not affect the display or process code pages.

```
#include <os2.h>

ULONG      ulReserved; /* Reserved. Must be 0. */
USHORT     pidCP;      /* Code page ID. */
HKBD       hkbd;       /* Reserved. Must be 0. */
APIRET     rc;          /* Return Code. */

rc = KbdSetCp(ulReserved, pidCP, hkbd);
```

KbdSetCp Parameter - ulReserved

ulReserved (ULONG) - input
Reserved. Must be 0.

KbdSetCp Parameter - pidCP

pidCP (USHORT) - input
Code page ID.

The code page ID must be the ID of a code page installed on the system or zero.

KbdSetCp Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdSetCp Return Value - rc

rc (APIRET) - returns
Return Code.

KbdSetCp returns one of the following values:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
448	ERROR_KBD_INVALID_CODEPAGE
504	ERROR_KBD_EXTENDED_SG

KbdSetCp - Parameters

ulReserved (ULONG) - input
Reserved. Must be 0.

pidCP (USHORT) - input
Code page ID.

The code page ID must be the ID of a code page installed on the system or zero.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return Code.

KbdSetCp returns one of the following values:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
448	ERROR_KBD_INVALID_CODEPAGE
504	ERROR_KBD_EXTENDED_SG

KbdSetCp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

KbdSetLayout

KbdSetLayout - Syntax

KbdSetLayout allows the keyboard layout to be changed. This change affects only the current session.

```
#include <os2.h>

PSZ      pszName; /* Keyboard layout name. */
HKBD     hkbd;    /* Reserved. Must be zero. */
APIRET   rc;      /* Return code. */

rc = KbdSetLayout(pszName, hkbd);
```

KbdSetLayout Parameter - pszName

pszName (PSZ) - input
Keyboard layout name.

The name of the keyboard layout. The length of the name cannot exceed 8 characters.

KbdSetLayout Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be zero.

KbdSetLayout returns one of the following values:

KbdSetLayout Return Value - rc

rc (APIRET) - returns
Return code.

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdSetLayout - Parameters

pszName (PSZ) - input

Keyboard layout name.

The name of the keyboard layout. The length of the name cannot exceed 8 characters.

hkbd (HKBD) - input
Reserved. Must be zero.

KbdSetLayout returns one of the following values:

rc (APIRET) - returns
Return code.

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdSetLayout - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

KbdSetLayoutUni

KbdSetLayoutUni - Syntax

[KbdSetLayout](#) allows the keyboard layout to be changed. This change affects only the current session.

```
#include <os2.h>

USHORT      *name; /* Keyboard layout name. */
HKBD        hkbd; /* Reserved. Must be 0. */
APIRET      rc;    /* Return code. */

rc = KbdSetLayoutUni(name, hkbd);
```

KbdSetLayoutUni Parameter - name

name (USHORT *) - input
Keyboard layout name.

Address of the unicode name of the keyboard layout. The name cannot exceed 8 UniChars in length.

KbdSetLayoutUni Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdSetLayoutUni Return Value - rc

rc (APIRET) - returns
Return code.

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdSetLayoutUni - Parameters

name (USHORT *) - input
Keyboard layout name.

Address of the unicode name of the keyboard layout. The name cannot exceed 8 UniChars in length.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

0	NO_ERROR
373	ERROR_KBD_PARAMETER
439	ERROR_KBD_INVALID_HANDLE

KbdSetLayoutUni - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

KbdSetStatus

KbdSetStatus - Syntax

KbdSetStatus sets the characteristics of the keyboard.

```
#define INCL_KBD
#include <os2.h>

PKBDINFO    pkbdinfo; /* Pointer to keyboard status. */
HKBD        hkbd;     /* Reserved. Must be 0. */
APIRET      rc;        /* Return code. */

rc = KbdSetStatus(pkbdinfo, hkbd);
```

KbdSetStatus Parameter - pkbdinfo

pkbdinfo (**PKBDINFO**) - output
Pointer to keyboard status.

A pointer to a **KBDINFO** structure in which the keyboard status is returned.

KbdSetStatus Parameter - hkbd

hkbd (**HKBD**) - input
Reserved. Must be 0.

KbdSetStatus Return Value - rc

rc (**APIRET**) - returns
Return code.

KbdSetStatus returns one of the following values:

0	NO_ERROR.
376	ERROR_KBD_INVALID_LENGTH
377	ERROR_KBD_INVALID_ECHO_MASK
378	ERROR_KBD_INVALID_INPUT_MASK
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdSetStatus - Parameters

pkbdinfo (**PKBDINFO**) - output
Pointer to keyboard status.

A pointer to a [KBDINFO](#) structure in which the keyboard status is returned.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdSetStatus returns one of the following values:

0	NO_ERROR.
376	ERROR_KBD_INVALID_LENGTH
377	ERROR_KBD_INVALID_ECHO_MASK
378	ERROR_KBD_INVALID_INPUT_MASK
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdSetStatus - Remarks

In the [KBDINFO](#) structure, the upper byte of *fsInterim* is the NLS shift state, and can be modified by KbdSetStatus.

KbdSetStatus - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

KbdStringIn

KbdStringIn - Syntax

KbdStringIn gets a string of keyboard input.

```
#define INCL_KBD
#include <os2.h>

PCH          pch;      /* Pointer to the character string buffer. */
PSTRINGINBUF pchin;    /* Pointer to buffer-length data. */
ULONG        Flag;     /* Wait flag. */
HKBD         hkbd;     /* Reserved. Must be 0. */
APIRET       rc;       /* Return code. */

rc = KbdStringIn(pch, pchin, Flag, hkbd);
```

KbdStringIn Parameter - pch

pch (PCH) - output
Pointer to the character string buffer.

KbdStringIn Parameter - pchin

pchin (PSTRINGINBUF) - in/out
Pointer to buffer-length data.

A pointer to the [STRINGINBUF](#) structure containing buffer-length data.

KbdStringIn Parameter - Flag

Flag (ULONG) - input
Wait flag.

0	IO_WAIT Wait for a keystroke if one is not available. The keystroke returned is removed from the queue.
1	IO_NOWAIT Return immediately, with or without a keystroke. If a keystroke is returned, remove it from the queue.
2	IO_PEEK Return immediately, with or without a keystroke. Do not remove the keystroke from the queue.
3	IO_PEEKWAIT Wait for a keystroke if one is not available. Return the keystroke but do not remove it from the queue.

KbdStringIn Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdStringIn Return Value - rc

rc (APIRET) - returns
Return code.

KbdStringIn returns one of the following values:

0	NO_ERROR
---	----------

373	ERROR_KBD_PARAMETER
375	ERROR_KBD_INVALID_IOWAIT
376	ERROR_KBD_INVALID_LENGTH
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
504	ERROR_KBD_EXTENDED_SG

KbdStringIn - Parameters

pch (PCH) - output
Pointer to the character string buffer.

pchin (PSTRINGINBUF) - in/out
Pointer to buffer-length data.

A pointer to the [STRINGINBUF](#) structure containing buffer-length data.

Flag (ULONG) - input
Wait flag.

0	IO_WAIT Wait for a keystroke if one is not available. The keystroke returned is removed from the queue.
1	IO_NOWAIT Return immediately, with or without a keystroke. If a keystroke is returned, remove it from the queue.
2	IO_PEEK Return immediately, with or without a keystroke. Do not remove the keystroke from the queue.
3	IO_PEEKWAIT Wait for a keystroke if one is not available. Return the keystroke but do not remove it from the queue.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdStringIn returns one of the following values:

0	NO_ERROR
373	ERROR_KBD_PARAMETER
375	ERROR_KBD_INVALID_IOWAIT
376	ERROR_KBD_INVALID_LENGTH
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
504	ERROR_KBD_EXTENDED_SG

KbdStringIn - Remarks

This function is retained only for compatibility. New code should not use it. [KbdGetConsole](#) is the preferred function for new code.

The contents and ending condition depend on the keyboard state:

ASCII	Only valid characters are placed in the input buffer. The function ends when a turnaround character is found or when the buffer is full. The turnaround character is not placed in the buffer.
BINARY	Both valid characters and extended keys are returned. The extended keys are returned as two-byte strings with a leading 0x00 or 0xe0 character. The function ends when the buffer is full.
TERM	Valid characters and escape sequences are placed in the input buffer. The function ends when the turnaround character is found, when an escape sequence is complete, or when the buffer is full. The turnaround character is placed in the buffer.

The maximum buffer length is 255 bytes.

The character strings can be optionally echoed on the display if echo mode is set. When echo is on, each character is echoed as it is read from the keyboard. Echo mode and binary mode are mutually exclusive. See [KbdGetStatus](#) and [KbdSetStatus](#) for more information.

The default input mode is ASCII. In ASCII mode, 2-byte character codes only return in complete form. An extended ASCII code is returned in a 2-byte string. The first byte is 0DH or E0H and the next byte is an extended code.

In input mode (binary or ASCII), the following returns can be set and retrieved with [KbdGetStatus](#) and [KbdSetStatus](#):

Turnaround Character
Echo Mode
Interim Character Flag
Shift State

The received input length is also used by the KbdStringIn line edit functions for re-displaying and entering a caller-specified string. On the next KbdStringIn call the received input length indicates the length of the input buffer that can be recalled by the user using the line-editing keys. A value of 0 inhibits the line-editing function for the current KbdStringIn request.

KbdStringIn - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

KbdXlate

KbdXlate - Syntax

KbdXlate translates scan codes with shift states into ASCII codes.

```
#define INCL_KBD
#include <os2.h>

PKBDKEYINFO    pKey; /* Pointer to character data. */
HKBD           hkbd; /* Reserved. Must be 0. */
APIRET         rc;   /* Return code. */

rc = KbdXlate(pKey, hkbd);
```

KbdXlate Parameter - pKey

pKey ([PKBDKEYINFO](#)) - in/out
Pointer to character data.

A pointer to a [KBDKEYINFO](#) structure containing character data.

KbdXlate Parameter - hkbd

hkbd (HKBD) - input
Reserved. Must be 0.

KbdXlate Return Value - rc

rc (APIRET) - returns
Return code.

KbdXlate returns one of the following values:

0	NO_ERROR.
376	ERROR_KDB_INVALID_LENGTH
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdXlate - Parameters

pKey ([PKBDKEYINFO](#)) - in/out
Pointer to character data.

A pointer to a [KBDKEYINFO](#) structure containing character data.

hkbd (HKBD) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

KbdXlate returns one of the following values:

0	NO_ERROR.
376	ERROR_KDB_INVALID_LENGTH
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
504	ERROR_KBD_EXTENDED_SG

KbdXlate - Remarks

KbdXlate is designed for conditions where the scan codes are known, but not the character. This must be used with care, and is not designed to be a substitute for the normal OS/2 keyboard translation functions.

The *resv* member of [KBDKEYINFO](#) must be maintained from call to call whenever an interim bit is set.

KbdXlate - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

Keyboard Data Types

This section describes the data types that are used with the Keyboard functions, and includes the following data types:

- [KBDHWID](#)
- [KBDINFO](#)
- [KBDKEYINFO](#)
- [STRINGINBUF](#)

KBDHWID

Keyboard hardware ID structure.

```
typedef struct _KBDHWID {  
    USHORT    cb;           /* Size, in bytes, of this data structure. */  
    USHORT    idKbd;        /* Keyboard hardware ID */  
    USHORT    idSecond;     /* Secondary ID */  
} KBDHWID;  
  
typedef KBDHWID *PKBDHWID;
```

KBDHWID Field - cb

cb (USHORT)

Size, in bytes, of this data structure.

On input, this field should contain the length of the Keyboard ID structure. The minimum input length value allowed is 2. On output, this field contains the actual number of bytes returned.

KBDHWID Field - idKbd

idKbd (USHORT)
Keyboard hardware ID

OS/2 supported keyboards and their hardware-generated IDs are as follows:

ID	Keyboard
0000H	Undetermined keyboard type
0001H	PC-AT Standard Keyboard
AB41H	101 Key Enhanced Keyboard
AB41H	102 Key Enhanced Keyboard
AB54H	88 and 89 Key Enhanced Keyboard
AB85H	122 Key Enhanced Keyboard

KBDHWID Field - idSecond

idSecond (USHORT)
Secondary ID

KBDINFO

Keyboard status data structure.

```
typedef struct _KBDINFO {  
    USHORT    cb;           /* Length, in bytes, of this data structure. */  
    USHORT    fsMask;       /* State mask. */  
    USHORT    chTurnAround; /* Turnaround character. */  
    USHORT    fsInterim;    /* Interim character state and NLS shift state. */  
    USHORT    fsState;      /* Current shift state. */  
} KBDINFO;  
  
typedef KBDINFO *PKBDINFO;
```

KBDINFO Field - cb

cb (USHORT)
Length, in bytes, of this data structure.

10 Only valid value.

KBDINFO Field - fsMask

fsMask (USHORT)

State mask.

The system state altered by this call. If bits 0 and 1 are off, the echo state of the system is not altered. If bits 2 and 3 are off, the binary and ASCII state of the system is not altered. If bits 0 and 1 are on, or if bits 2 and 3 are on, the function returns an error. If binary mode is set, echo is ignored.

Bits 15-9	Reserved.
Bit 8	Shift return is on.
Bit 7	Length of the turn-around character (meaningful only if bit 6 is on).
Bit 6	Turn-around character is modified.
Bit 5	Interim character flags are modified.
Bit 4	Shift state is modified.
Bit 3	ASCII mode is on.
Bit 2	Binary mode is on.
Bit 1	Echo off.
Bit 0	Echo on.

KBDINFO Field - chTurnAround

chTurnAround (USHORT)

Turnaround character.

Definition of the turn-around character. In ASCII and extended-ASCII format, the turn-around character is defined as the carriage return. In ASCII format only, the turn-around character is defined in the low-order byte.

KBDINFO Field - fsInterim

fsInterim (USHORT)

Interim character state and NLS shift state.

Bits 15-8	NLS shift state.
	The meaning of the NLS shift varies by language. The following bits are defined to access this data:
	NLSS_NLS1 (0x01) - Fullwidth, National layer
	NLSS_NLS2 (0x02) - Katakana, JAMO phonetic
	NLSS_NLS3 (0x04) - Hiragana, Hangeul, TsangJye
	NLSS_APPL (0x10) - Application bit
	NLSS_NLS4 (0x40) - Romanji, HanjaCsr
	NLSS_KANJI (0x80) - Kanji, Hanji
Bit 7	Interim character flag is on.
Bit 6	Reserved.

Bit 5	Application requested immediate conversion.
Bits 4-0	Reserved.

KBDINFO Field - fsState

fsState (USHORT)
Current shift state.

Bit 15	SysReq key down.
Bit 14	CapsLock key down.
Bit 13	NumLock key down.
Bit 12	ScrollLock key down.
Bit 11	Right Alt key down.
Bit 10	Right Ctrl key down.
Bit 9	Left Alt key down.
Bit 8	Left Ctrl key down.
Bit 7	Insert on.
Bit 6	CapsLock on.
Bit 5	NumLock on.
Bit 4	ScrollLock on.
Bit 3	Either Alt key down.
Bit 2	Either Ctrl key down.
Bit 1	Left Shift key down.
Bit 0	Right Shift key down.

KBDKEYINFO

The Character data structure of the API function KbdCharIn.

```
typedef struct _KBDKEYINFO {
    USHORT    ucUniChar; /* Unicode character. */
    USHORT    chChar;    /* ASCII Character code. */
    UCHAR     chScan;    /* Code received for the keyboard. */
    UCHAR     fbStatus;   /* State of the keystroke event flag. */
    USHORT    fsState;   /* Shift key status flag. */
    USHORT    VKey;      /* Virtual key. */
    USHORT    resv;      /* Reserved, must be zero. */
    ULONG     time;      /* Time stamp in milliseconds. */
} KBDKEYINFO;

typedef KBDKEYINFO *PKBDKEYINFO;
```

KBDKEYINFO Field - ucUniChar

ucUniChar (USHORT)
Unicode character.

KBDKEYINFO Field - chChar

chChar (USHORT)
ASCII Character code.

The scan code received from the keyboard is translated to the ASCII character code.

KBDKEYINFO Field - chScan

chScan (UCHAR)
Code received for the keyboard.

Scan code received from the keyboard is translated to the ASCII character code.

KBDKEYINFO Field - fbStatus

fbStatus (UCHAR)
State of the keystroke event flag.

Bits 7-6	Has the following values:	
	00	Undefined.
	01	Final character; interim character flag is turned <i>off</i> .
	10	Interim character.
	11	Final character; interim character flag is turned <i>on</i> .
Bit 5	If set to 1, immediate conversion requested.	
Bits 4-2	Reserved.	
Bit 1	Has the following values:	
	0	Scan code is a character
	1	Scan code is not a character; instead it is an extended key code from the keyboard.
Bit 0	If set to 1, shift status returned without a character.	

KBDKEYINFO Field - fsState

fsState (USHORT)
Shift key status flag.

Values are:

Bit 15	SysReq key down
Bit 14	Caps Lock key down
Bit 13	NumLock key down
Bit 12	Scroll Lock key down
Bit 11	Right Alt key down
Bit 10	Right Ctrl key down
Bit 9	Left Alt key down
Bit 8	Left Ctrl key down
Bit 7	Insert <i>on</i>
Bit 6	Caps Lock <i>on</i>
Bit 5	NumLock <i>on</i> .
Bit 4	Scroll Lock <i>on</i>
Bit 3	Either Alt key down
Bit 2	Either Ctrl key down
Bit 1	Left Shift key down
Bit 0	Right Shift key down

KBDKEYINFO Field - VKey

VKey (USHORT)
Virtual key.

KBDKEYINFO Field - resv

resv (USHORT)
Reserved, must be zero.

The upper byte within resv is the NLS Shift state.

KBDKEYINFO Field - time

time (ULONG)
Time stamp in milliseconds.

Time stamp indicating when a key was pressed. It is specified in milliseconds from the time the system was started.

STRINGINBUF

The buffer length data structure.

```
typedef struct _STRINGINBUF {  
    ULONG      cb;          /* Length, of the input buffer, in bytes. */  
    ULONG      cchIn;       /* Number of bytes returned. */  
};
```



```
} STRINGINBUF;  
  
typedef STRINGINBUF *PSTRINGINBUF;
```

STRINGINBUF Field - cb

cb (ULONG)
Length, of the input buffer, in bytes.

STRINGINBUF Field - cchIn

cchIn (ULONG)
Number of bytes returned.

Mouse

This chapter describes all of the 32-bit Mouse functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

The chapter includes the following sections:

- [Mouse Functions](#)
 - [Mouse Data Types](#)
-

Mouse Functions

The Mouse functions described in this section are:

- `apiref refid=moudraw.`
- `apiref refid=mouflush.`
- `apiref refid=moudevst.`
- `apiref refid=mougtmsk.`
- `apiref refid=mounumbt.`
- `apiref refid=mounummk.`
- `apiref refid=mougnqe.`
- `apiref refid=mougpos.`
- `apiref refid=mougposh.`
- `apiref refid=mougscf.`
- `apiref refid=mougtrh.`
- `apiref refid=mourevq.`
- `apiref refid=mourptr.`
- `apiref refid=mousetds.`
- `apiref refid=mousetem.`
- `apiref refid=mouspos.`
- `apiref refid=mousetps.`
- `apiref refid=mousscf.`
- `apiref refid=mousthr.`

MouDrawPtr

MouDrawPtr - Syntax

MouDrawPtr notifies the mouse device driver that an area previously restricted by [MouRemovePtr](#) is now available to the mouse device driver when drawing pointer images.

```
#define INCL_MOU
#include <os2.h>

HMOU      DeviceHandle; /* Mouse device handle. */
APIRET    rc;           /* Return code. */

rc = MouDrawPtr(DeviceHandle);
```

MouDrawPtr Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Must be 0.

MouDrawPtr Return Value - rc

rc (APIRET) - returns
Return code.

MouDrawPtr returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouDrawPtr - Parameters

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouDrawPtr returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouDrawPtr - Remarks

MouDrawPtr is required to begin a pointer-image drawing session. At session start, the collision area (the area restricted from pointer-image drawing) is defined as the size of the display.

After a call to MouDrawPtr, [MouRemovePtr](#) can be used to define a new collision area. A subsequent call to MouDrawPtr will cancel the collision area.

MouDrawPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouFlushQue

MouFlushQue - Syntax

MouFlushQue directs the mouse device driver to empty the mouse-event queue.

```
#define INCL_MOU
#include <os2.h>

HMOU      DeviceHandle; /* Mouse device handle. */
APIRET    rc;           /* Return code. */

rc = MouFlushQue(DeviceHandle);
```

MouFlushQue Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Must be 0.

MouFlushQue Return Value - rc

rc (APIRET) - returns
Return code.

MouFlushQue returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouFlushQue - Parameters

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouFlushQue returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouFlushQue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouGetDevStatus

MouGetDevStatus - Syntax

MouDevStatus returns status flags for the installed mouse device driver.

```
#define INCL_MOUSE
#include <os2.h>

PULONG DeviceStatus; /* Current status flags. */
HMOU DeviceHandle; /* Mouse device handle. */
APIRET rc; /* Return code. */

rc = MouGetDevStatus(DeviceStatus, DeviceHandle);
```

MouGetDevStatus Parameter - DeviceStatus

DeviceStatus (PULONG) - input
Current status flags.

Address of the current status-flag settings for the installed mouse device driver.

The return value is a 4-byte set of bit flags.

Bit	Description
31-10	Reserved. Set to 0.
9	If set, mouse data is returned in mickeys, not pels.
8	If set, the drawing operations for the pointer draw routine are disabled.
7-4	Reserved. Set to 0.
3	If set, the pointer draw routine is disabled by an unsupported mode.
2	If set, a flush is in progress.
1	If set, a block-read is in progress.
0	If set, the event queue is busy with I/O.

MouGetDevStatus Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Set to 0.

MouGetDevStatus Return Value - rc

rc (APIRET) - returns
Return code.

MouDevStatus returns one of the following values:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOUSE_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetDevStatus - Parameters

DeviceStatus (PULONG) - input
Current status flags.

Address of the current status-flag settings for the installed mouse device driver.

The return value is a 4-byte set of bit flags.

Bit	Description
31-10	Reserved. Set to 0.
9	If set, mouse data is returned in mickeys, not pels.
8	If set, the drawing operations for the pointer draw routine are disabled.
7-4	Reserved. Set to 0.
3	If set, the pointer draw routine is disabled by an unsupported mode.
2	If set, a flush is in progress.
1	If set, a block-read is in progress.
0	If set, the event queue is busy with I/O.

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Set to 0.

rc (APIRET) - returns
Return code.

MouDevStatus returns one of the following values:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOUSE_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetDevStatus - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouGetEventMask

MouGetEventMask - Syntax

MouGetEventMask returns the current value of the mouse-event queue mask.

```
#define INCL_MOU
#include <os2.h>

PULONG      EventMask;      /* Event mask word. */
HMOU        DeviceHandle;   /* Mouse device handle. */
APIRET      rc;             /* Return code. */

rc = MouGetEventMask(EventMask, DeviceHandle);
```

MouGetEventMask Parameter - EventMask

EventMask (PULONG) - input
Event mask word.

Address in application storage where the event mask of the current mouse device driver is returned to the caller by the mouse device driver.

The EventMask is set by [MouSetEventMask](#) and has the following definition:

Bit	Description
31-7	Reserved. Set to 0.
6	Report button-3 press/release events, without mouse motion.
5	Report button-3 press/release events, with mouse motion.
4	Report button-2 press/release events, without mouse motion.
3	Report button-2 press/release events, with mouse motion.
2	Report button-1 press/release events, without mouse motion.
1	Report button-1 press/release events, with mouse motion.
0	Report mouse motion events with no button press/release events.

MouGetEventMask Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Set to 0.

MouGetEventMask Return Value - rc

rc (APIRET) - returns
Return code.

MouGetEventMask returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetEventMask - Parameters

EventMask (PULONG) - input
Event mask word.

Address in application storage where the event mask of the current mouse device driver is returned to the caller by the mouse device driver.

The EventMask is set by [MouSetEventMask](#) and has the following definition:

Bit	Description
31-7	Reserved. Set to 0.
6	Report button-3 press/release events, without mouse motion.
5	Report button-3 press/release events, with mouse motion.
4	Report button-2 press/release events, without mouse motion.
3	Report button-2 press/release events, with mouse motion.
2	Report button-1 press/release events, without mouse motion.
1	Report button-1 press/release events, with mouse motion.
0	Report mouse motion events with no button press/release events.

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Set to 0.

rc (APIRET) - returns
Return code.

MouGetEventMask returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetEventMask - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouGetNumButtons

MouGetNumButtons - Syntax

MouGetNumButtons returns the number of buttons supported on the installed mouse driver.

```
#define INCL_MOUSE
#include <os2.h>

PULONG    NumberOfButtons; /* Number of mouse buttons. */
HMOU      DeviceHandle;    /* Mouse device handle. */
APIRET    rc;              /* Return code. */

rc = MouGetNumButtons(NumberOfButtons, DeviceHandle);
```

MouGetNumButtons Parameter - NumberOfButtons

NumberOfButtons (PULONG) - output
Number of mouse buttons.

Address where the number of physical buttons is to be returned.

The return values for the number of buttons supported are:

	Value	Definition
1		One mouse button.
2		Two mouse buttons.
3		Three mouse buttons.

MouGetNumButtons Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Mouse device handle.

Reserved. Set to 0.

MouGetNumButtons Return Value - rc

rc (APIRET) - returns
Return code.

MouGetNumButtons returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetNumButtons - Parameters

NumberOfButtons (PULONG) - output

Number of mouse buttons.

Address where the number of physical buttons is to be returned.

The return values for the number of buttons supported are:

Value	Definition
1	One mouse button.
2	Two mouse buttons.
3	Three mouse buttons.

DeviceHandle (HMOU) - input

Mouse device handle.

Reserved. Set to 0.

rc (APIRET) - returns

Return code.

MouGetNumButtons returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetNumButtons - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouGetNumMickeys

MouGetNumMickeys - Syntax

MouGetNumMickeys returns the number of mickeys per centimeter for the installed mouse driver.

```

#define INCL_MOUSE
#include <os2.h>

PULONG    NumberOfMickey; /* Number of mickeys per centimeter. */
HMOU      DeviceHandle; /* Mouse device handle */
APIRET    rc; /* Return code. */

rc = MouGetNumMickey(NumberOfMickey, DeviceHandle);

```

MouGetNumMickey Parameter - NumberOfMickey

NumberOfMickey (PULONG) - input
Number of mickeys per centimeter.

Address of the number of physical mouse motion units. Mouse motion units are reported in mickeys per centimeter. This value is constant, and is based on the mouse device attached.

MouGetNumMickey Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Mouse device handle

Reserved. Must be 0.

MouGetNumMickey Return Value - rc

rc (APIRET) - returns
Return code.

MouGetNumMickey returns one of the following values:

0	NO_ERROR
466	ERROR_MOUSE_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetNumMickey - Parameters

NumberOfMickey (PULONG) - input
Number of mickeys per centimeter.

Address of the number of physical mouse motion units. Mouse motion units are reported in mickeys per centimeter. This value is constant, and is based on the mouse device attached.

DeviceHandle (HMOU) - input
Mouse device handle

Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouGetNumMickey returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetNumMickey - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouGetNumQueue

MouGetNumQueue - Syntax

MouGetNumQueue returns the current status for the mouse device-driver event queue.

```
#define INCL_MOU
#include <os2.h>

PMOUSEINFO    QueDataRecord; /* Pointer to the mouse queue data structure. */
HMOU          DeviceHandle;   /* Reserved. Must be 0. */
APIRET        rc;             /* Return code. */

rc = MouGetNumQueue(QueDataRecord, DeviceHandle);
```

MouGetNumQueue Parameter - QueDataRecord

QueDataRecord ([PMOUSEINFO](#)) - output
Pointer to the mouse queue data structure.

MouGetNumQueue Parameter - DeviceHandle

DeviceHandle (HMOU) - input

Reserved. Must be 0.

MouGetNumQueEl Return Value - rc

rc (APIRET) - returns
Return code.

MouGetNumQueEl returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetNumQueEl - Parameters

QueDataRecord ([PMOUQUEINFO](#)) - output
Pointer to the mouse queue data structure.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouGetNumQueEl returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetNumQueEl - Remarks

The maximum number of queue elements returned in *cmaxEvents* is established during mouse device-driver configuration.

MouGetNumQueEl - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouGetPtrPos

MouGetPtrPos - Syntax

MouGetPtrPos queries the mouse driver to determine the current row-and-column coordinate position of the mouse pointer.

```
#define INCL_MOU
#include <os2.h>

PPTRLOC    PtrPos;      /* Pointer to the mouse pointer data structure. */
HMOU       DeviceHandle; /* Reserved. Must be 0. */
APIRET     rc;          /* Return code. */

rc = MouGetPtrPos(PtrPos, DeviceHandle);
```

MouGetPtrPos Parameter - PtrPos

PtrPos (**PPTRLOC**) - output
Pointer to the mouse pointer data structure.

MouGetPtrPos Parameter - DeviceHandle

DeviceHandle (**HMOU**) - input
Reserved. Must be 0.

MouGetPtrPos Return Value - rc

rc (**APIRET**) - returns
Return code.

MouGetPtrPos returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetPtrPos - Parameters

PtrPos (**PPTRLOC**) - output

Pointer to the mouse pointer data structure.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouGetPtrPos returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetPtrPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouGetPtrShape

MouGetPtrShape - Syntax

MouGetPtrShape gets the pointer shape for the session.

```
#define INCL_MOU
#include <os2.h>

PVOID      PtrBuffer;      /* Pointer shape buffer. */
PPTRSHAPE  PtrDefRec;      /* Pointer definition structure. */
HMOU       DeviceHandle;    /* Reserved. Must be 0. */
APIRET     rc;              /* Return code. */

rc = MouGetPtrShape(PtrBuffer, PtrDefRec,
    DeviceHandle);
```

MouGetPtrShape Parameter - PtrBuffer

PtrBuffer (PVOID) - output
Pointer shape buffer.

Address of a buffer containing the bit image used as the pointer shape for the session. The buffer consists of AND and XOR pointer masks.

For CGA-compatible text modes (0, 1, 2, and 3) the following describes the AND and XOR pointer mask bit definitions for each

character cell of the masks. Bit values are*colon.

15	Blinking
14-12	Background color
11	Intensity
10-8	Foreground color
7-0	Character

MouGetPtrShape Parameter - PtrDefRec

PtrDefRec (**PPTRSHAPE**) - in/out
Pointer definition structure.

Contains information about the pointer shape.

MouGetPtrShape Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouGetPtrShape Return Value - rc

rc (APIRET) - returns
Return code.

MouGetPtrShape returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetPtrShape - Parameters

PtrBuffer (PVOID) - output
Pointer shape buffer.

Address of a buffer containing the bit image used as the pointer shape for the session. The buffer consists of AND and XOR pointer masks.

For CGA-compatible text modes (0, 1, 2, and 3) the following describes the AND and XOR pointer mask bit definitions for each character cell of the masks. Bit values are*colon.

15	Blinking
14-12	Background color
11	Intensity
10-8	Foreground color

PtrDefRec ([PPTRSHAPE](#)) - in/out
Pointer definition structure.

Contains information about the pointer shape.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouGetPtrShape returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetPtrShape - Remarks

The application passes a parameter list with the same meaning as defined for [MouSetPtrShape](#) to the mouse device driver. The mouse device driver copies the parameters that describe the pointer shape and attributes into the pointer definition control block pointed to by the *PtrDefRec* parameter. The *cb* field must contain the size in bytes of the application buffer where the device driver is to insert the sessions pointer image. All other fields are returned to the application by MouGetPtrShape.

If the buffer size is insufficient, the *cb* field contains the actual size in bytes of the returned pointer image.

The pointer shape may be set by the application with MouSetPtrShape or may be the default image provided by the installed Pointer Device Driver.

MouGetPtrShape - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouGetScaleFact

MouGetScaleFact - Syntax

MouGetScaleFact returns a pair of one-word scaling factors for the current mouse device.

```
#define INCL_MOU
#include <os2.h>
```

```

PSCALEFACT    ScaleStruct;    /* Pointer to a control-block structure. */
HMOU          DeviceHandle;    /* Reserved. Must be 0. */
APIRET        rc;              /* Return code. */

rc = MouGetScaleFact(ScaleStruct, DeviceHandle);

```

MouGetScaleFact Parameter - ScaleStruct

ScaleStruct ([PSCALEFACT](#)) - output
 Pointer to a control-block structure.

Address of the control-block structure that contains the current row and column coordinate scaling factors. The scaling factors must be greater than or equal to 1, and less than or equal to (32K - 1).

MouGetScaleFact Parameter - DeviceHandle

DeviceHandle (HMOU) - input
 Reserved. Must be 0.

MouGetScaleFact Return Value - rc

rc (APIRET) - returns
 Return code.

MouGetScaleFact returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetScaleFact - Parameters

ScaleStruct ([PSCALEFACT](#)) - output
 Pointer to a control-block structure.

Address of the control-block structure that contains the current row and column coordinate scaling factors. The scaling factors must be greater than or equal to 1, and less than or equal to (32K - 1).

DeviceHandle (HMOU) - input
 Reserved. Must be 0.

rc (APIRET) - returns
 Return code.

MouGetScaleFact returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetScaleFact - Remarks

The units of the scale factor depend on the mode of the display screen for the session. If the screen is operating in text mode, the scaling units are relative to characters. If the screen is operating in graphics mode, the scaling units are relative to pels.

MouGetScaleFact - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouGetThreshold

MouGetThreshold - Syntax

MouGetThreshold returns the current threshold values.

```
#define INCL_MOU
#include <os2.h>

PTHRESHOLD    Threshold;    /* Pointer threshold structure. */
HMOU          DeviceHandle; /* Reserved. Must be 0. */
APIRET        rc;           /* Return code. */

rc = MouGetThreshold(Threshold, DeviceHandle);
```

MouGetThreshold Parameter - Threshold

Threshold ([PTHRESHOLD](#)) - input
Pointer threshold structure.

Address of the control block structure that contains the threshold structure.

MouGetThreshold Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouGetThreshold Return Value - rc

rc (APIRET) - returns
Return code.

MouGetThreshold returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetThreshold - Parameters

Threshold ([PTHRESHOLD](#)) - input
Pointer threshold structure.

Address of the control block structure that contains the threshold structure.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouGetThreshold returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouGetThreshold - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouReadEventQue

MouReadEventQue - Syntax

MouReadEventQue reads an event from the mouse-device FIFO event queue and places it in a structure provided by the application.

```
#define INCL_MOUSE
#include <os2.h>

PMOUEVENTINFO    EventMask;    /* Pointer to the mouse-event queue. */
PULONG           Wait;         /* Wait flag. */
HMOU              DeviceHandle; /* Reserved. Must be 0. */
APIRET           rc;           /* Return code. */

rc = MouReadEventQue(EventMask, Wait, DeviceHandle);
```

MouReadEventQue Parameter - EventMask

EventMask (PMOUEVENTINFO) - output
Pointer to the mouse-event queue.

Address of the mouse-event queue data.

MouReadEventQue Parameter - Wait

Wait (PULONG) - input
Wait flag.

A pointer to a ULONG containing the action to take when MouReadEventQue is issued and the mouse event queue is empty. If the mouse event queue is not empty, this parameter is not examined by the mouse support.

May be one of the following values:

- | | |
|---|--|
| 0 | MOU_NOWAIT
Do not wait for data on an empty queue (return a NULL record). |
| 1 | MOU_WAIT
Wait for data on an empty queue. |
-

MouReadEventQue Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouReadEventQue Return Value - rc

rc (APIRET) - returns
Return code.

MouReadEventQue returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
393	ERROR_MOUSE_NO_DATA
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouReadEventQue - Parameters

EventMask (PMOUEVENTINFO) - output
Pointer to the mouse-event queue.

Address of the mouse-event queue data.

Wait (PULONG) - input
Wait flag.

A pointer to a ULONG containing the action to take when MouReadEventQue is issued and the mouse event queue is empty. If the mouse event queue is not empty, this parameter is not examined by the mouse support.

May be one of the following values:

0	MOU_NOWAIT Do not wait for data on an empty queue (return a NULL record).
1	MOU_WAIT Wait for data on an empty queue.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouReadEventQue returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
393	ERROR_MOUSE_NO_DATA
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouReadEventQue - Remarks

The types of queued events are directly affected by the current value of the event mark. [MouSetEventMask](#) is used to indicate the types of events desired, and [MouGetEventMask](#) is used to query the current value of the mask. Refer to these functions for more information about the

masking of events.

MouReadEventQue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouRemovePtr

MouRemovePtr - Syntax

MouRemovePtr notifies the mouse device driver that the area defined by the passed parameters is for the exclusive use of the application. This area is defined as the collision area and is not available to the mouse device driver when drawing pointer images.

```
#define INCL_MOU
#include <os2.h>

PNOPTRRECT    PtrArea;        /* Pointer shape collision area. */
HMOU          DeviceHandle;    /* Reserved. Must be 0. */
APIRET        rc;              /* Return code. */

rc = MouRemovePtr(PtrArea, DeviceHandle);
```

MouRemovePtr Parameter - PtrArea

PtrArea ([PNOPTRRECT](#)) - input
Pointer shape collision area.

The address of the pointer shape collision area structure.

MouRemovePtr Parameter - DeviceHandle

DeviceHandle ([HMOU](#)) - input
Reserved. Must be 0.

MouRemovePtr Return Value - rc

rc (APIRET) - returns
Return code.

MouRemovePtr returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouRemovePtr - Parameters

PtrArea ([PNOPTRECT](#)) - input
Pointer shape collision area.

The address of the pointer shape collision area structure.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouRemovePtr returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouRemovePtr - Remarks

MouRemovePtr can be issued by any process in the session. However, only one collision area at a time is active. Each call to MouRemovePtr has the effect of resetting the collision area to the location and area specified by the current call.

If the logical pointer position is outside of the collision area specified by the latest MouRemovePtr call, the pointer image is drawn.

The A [MouDrawPtr](#) call effectively cancels a MouRemovePtr call, allowing the pointer to be drawn anywhere on the screen until the next MouRemovePtr call is issued.

MouRemovePtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouSetDevStatus

MouSetDevStatus - Syntax

MouSetDevStatus sets the mouse device driver status flags for the installed mouse device driver.

```
#define INCL_MOUSE
#include <os2.h>

PULONG    DeviceStatus; /* Status flags. */
HMOU      DeviceHandle; /* Reserved. Must be 0. */
APIRET    rc;           /* Return code. */

rc = MouSetDevStatus(DeviceStatus, DeviceHandle);
```

MouSetDevStatus Parameter - DeviceStatus

DeviceStatus (PULONG) - input
Status flags.

The passed parameter is a 4-byte set of flags. Only the high byte has meaning.

Bit	Description
31-10	Reserved. Set to 0.
9	If set, mouse data is returned in mickeys, not pels.
8	If set, the drawing operations for pointer draw routine are disabled.
7-0	Reserved. Set to 0.

MouSetDevStatus Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouSetDevStatus Return Value - rc

rc (APIRET) - returns
Return code.

MouSetDevStatus returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARAMS
466	ERROR_MOUSE_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetDevStatus - Parameters

DeviceStatus (PULONG) - input
Status flags.

The passed parameter is a 4-byte set of flags. Only the high byte has meaning.

Bit	Description
31-10	Reserved. Set to 0.
9	If set, mouse data is returned in mickeys, not pels.
8	If set, the drawing operations for pointer draw routine are disabled.
7-0	Reserved. Set to 0.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouSetDevStatus returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARAMS
466	ERROR_MOUSE_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetDevStatus - Remarks

MouSetDevStatus is the complement to [MouGetDevStatus](#). However, not all status flags can be set with MouSetDevStatus. Only the flags corresponding to the following functions can be modified:

- Return data in mickeys.
 - Normally, mouse data is returned to the application with the absolute display mode coordinates of the pointer image position on the display screen. When this status flag is set, mouse data is returned in relative mickeys, a unit of mouse movement.
- Don't call pointer draw device.
 - Normally, the pointer draw device driver is called for all drawing operations. When this status flag is set, the mouse device driver does not call the pointer draw device driver. The application must draw any required pointer image on the screen.

At session initialization, the device status is set to 0.

MouSetDevStatus - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouSetEventMask

MouSetEventMask - Syntax

MouSetEventMask assigns a new event mask to the current mouse device driver.

```
#define INCL_MOUSE
#include <os2.h>

PULONG      EventMask;      /* Mouse device event mask pointer. */
HMOU        DeviceHandle;   /* Reserved. Must be 0. */
APIRET      rc;             /* Return code. */

rc = MouSetEventMask(EventMask, DeviceHandle);
```

MouSetEventMask Parameter - EventMask

EventMask (PULONG) - input
Mouse device event mask pointer.

The EventMask bit values are:

Bit	Description
31-7	Reserved. Set to 0.
6	Report button 3 press/release events, without mouse motion.
5	Report button 3 press/release events, with mouse motion.
4	Report button 2 press/release events, without mouse motion.
3	Report button 2 press/release events, with mouse motion.
2	Report button 1 press/release events, without mouse motion.
1	Report button 1 press/release events, with mouse motion.

MouSetEventMask Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouSetEventMask Return Value - rc

rc (APIRET) - returns
Return code.

MouSetEventMask returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetEventMask - Parameters

EventMask (PULONG) - input
Mouse device event mask pointer.

The EventMask bit values are:

Bit	Description
31-7	Reserved. Set to 0.
6	Report button 3 press/release events, without mouse motion.
5	Report button 3 press/release events, with mouse motion.
4	Report button 2 press/release events, without mouse motion.
3	Report button 2 press/release events, with mouse motion.
2	Report button 1 press/release events, without mouse motion.
1	Report button 1 press/release events, with mouse motion.
0	Mouse motion events with no button press/release events.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouSetEventMask returns one of the following values:

0	NO_ERROR
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetEventMask - Remarks

Setting a bit in the event mask means that the associated event is reported on the mouse FIFO event queue.

At session initialization, the event mask is set to report all events.

MouSetEventMask - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouSetPtrPos

MouSetPtrPos - Syntax

MouSetPtrPos directs the mouse driver to set a new row-and-column coordinate position for the mouse pointer.

```
#define INCL_MOU
#include <os2.h>

PPTRLOC    PtrPos;        /* Pointer position. */
HMOU       DeviceHandle;  /* Reserved. Must be 0. */
APIRET     rc;            /* Return code. */

rc = MouSetPtrPos(PtrPos, DeviceHandle);
```

MouSetPtrPos Parameter - PtrPos

PtrPos ([PPTRLOC](#)) - input
 Pointer position.

Address of the mouse pointer position structure.

MouSetPtrPos Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouSetPtrPos Return Value - rc

rc (APIRET) - returns
Return code.

MouSetPtrPos returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetPtrPos - Parameters

PtrPos ([PPTRLOC](#)) - input
Pointer position.

Address of the mouse pointer position structure.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

MouSetPtrPos returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetPtrPos - Remarks

The application must ensure that the specified coordinate position conforms to the current display mode orientation for the session. Pel values must be used for graphics modes, and character values must be used for text modes.

This function has no effect on the definition of the display's current collision area specified in [MouDrawPtr](#). If the mouse pointer image is directed into a defined collision area, the pointer image is not drawn until either the pointer is moved outside the collision area, or the collision area is released by [MouDrawPtr](#).

At session initialization, the pointer is set to the center of the screen but is not drawn, because the collision area is set to the entire screen.

MouSetPtrPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouSetPtrShape

MouSetPtrShape - Syntax

MouSetPtrShape sets the pointer shape and size to be used as the mouse device driver pointer image for all applications in a session.

```
#define INCL_MOU
#include <os2.h>

PBYTE      PtrBuffer;      /* Pointer-shape buffer. */
PPTRSHAPE  PtrDefRec;      /* Pointer definition record. */
HMOU       DeviceHandle;    /* Reserved. Must be 0. */
APIRET     rc;              /* Return code. */

rc = MouSetPtrShape(PtrBuffer, PtrDefRec,
    DeviceHandle);
```

MouSetPtrShape Parameter - PtrBuffer

PtrBuffer (PBYTE) - input
Pointer-shape buffer.

Address of a buffer containing the bit image used as the pointer shape for the session. The buffer consists of AND and XOR pointer masks.

For CGA-compatible text modes (0, 1, 2, and 3) the following describes the AND and XOR pointer mask bit definitions for each character cell of the masks. Bit values are*colon.

15	Blinking
14-12	Background color
11	Intensity
10-8	Foreground color
7-0	Character

MouSetPtrShape Parameter - PtrDefRec

PtrDefRec (**PPTRSHAPE**) - input
Pointer definition record.

Address of the structure where the application stores the necessary data for the pointer draw device driver to build a row-by-column image for each bit plane for the current display mode.

MouSetPtrShape Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouSetPtrShape Return Value - rc

rc (APIRET) - returns
Return code.

MouSetPtrShape returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetPtrShape - Parameters

PtrBuffer (PBYTE) - input
Pointer-shape buffer.

Address of a buffer containing the bit image used as the pointer shape for the session. The buffer consists of AND and XOR pointer masks.

For CGA-compatible text modes (0, 1, 2, and 3) the following describes the AND and XOR pointer mask bit definitions for each character cell of the masks. Bit values are*colon.

15	Blinking
14-12	Background color
11	Instensity
10-8	Foreground color
7-0	Character

PtrDefRec (**PPTRSHAPE**) - input
Pointer definition record.

Address of the structure where the application stores the necessary data for the pointer draw device driver to build a row-by-column image for each bit plane for the current display mode.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns

Return code.

MouSetPtrShape returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetPtrShape - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

MouSetScaleFact

MouSetScaleFact - Syntax

MouSetScaleFact assigns a new set of scaling factors to the current mouse device driver.

```
#define INCL_MOU
#include <os2.h>

PSCALEFACT    ScaleStruct;    /* Address of scaling factors. */
HMOU          DeviceHandle;    /* Reserved. Must be 0. */
APIRET        rc;             /* Return code */

rc = MouSetScaleFact(ScaleStruct, DeviceHandle);
```

MouSetScaleFact Parameter - ScaleStruct

ScaleStruct ([PSCALEFACT](#)) - input
Address of scaling factors.

The address of the control block structure that contains the current row-and-column coordinate scaling factors. The scaling factors must be greater than or equal to 1 and less than or equal to (32K-1).

MouSetScaleFact Parameter - DeviceHandle

DeviceHandle (HMOU) - input
Reserved. Must be 0.

MouSetScaleFact Return Value - rc

rc (APIRET) - returns
Return code

MouSetScaleFact returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetScaleFact - Parameters

ScaleStruct ([PSCALEFACT](#)) - input
Address of scaling factors.

The address of the control block structure that contains the current row-and-column coordinate scaling factors. The scaling factors must be greater than or equal to 1 and less than or equal to (32K-1).

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code

MouSetScaleFact returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetScaleFact - Remarks

MouSetScaleFact sets the mickey-to-pixel ratio for mouse motion. The row scale and column scale ratios specify a number of mickeys for each 8 pixels. The default value for the row scale is 16 mickeys for each 8 pixels. The default value for the column scale is 8 mickeys to 8 pixels.

The number of pixels moved does not have to be one-to-one for the number of mickeys the mouse moves. The scaling factor defines a sensitivity for the mouse that is a ratio of the number of mickeys required to move the cursor 8 pixels on the screen. The sensitivity determines at what rate the cursor moves on the screen.

MouSetScaleFact - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MouSetThreshold

MouSetThreshold - Syntax

MouSetThreshold assigns a new set of threshold values to the current mouse device driver.

```
#define INCL_MOU
#include <os2.h>

PTHRESHOLD    Threshold;    /* Threshold structure. */
HMOU           DeviceHandle; /* Reserved. Must be 0. */
APIRET        rc;           /* Return code */

rc = MouSetThreshold(Threshold, DeviceHandle);
```

MouSetThreshold Parameter - Threshold

Threshold ([PTHRESHOLD](#)) - input
Threshold structure.

The address of the control block structure that contains the threshold structure.

MouSetThreshold Parameter - DeviceHandle

DeviceHandle ([HMOU](#)) - input
Reserved. Must be 0.

MouSetThreshold Return Value - rc

rc ([APIRET](#)) - returns
Return code

MouSetThreshold returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMs
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetThreshold - Parameters

Threshold ([PTHRESHOLD](#)) - input
Threshold structure.

The address of the control block structure that contains the threshold structure.

DeviceHandle (HMOU) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code

MouSetThreshold returns one of the following values:

0	NO_ERROR
387	ERROR_MOUSE_INV_PARMs
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE

MouSetThreshold - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

Mouse Data Types

This section describes the data types that are used with the Mouse functions, and includes the following data types:

- [MOUEVENTINFO](#)
- [MOUQUEINFO](#)
- [NOPTRRECT](#)
- [PTRLOC](#)
- [PTRSHAPE](#)
- [SCALEFACT](#)
- [THRESHOLD](#)

MOUEVENTINFO

Mouse event queue data structure.

```
typedef struct _MOUEVENTINFO {
    ULONG     fs;      /* Mouse state. */
    LONG      row;     /* Horizontal position. */
    LONG      col;     /* Vertical position. */
    ULONG     time;    /* Timestamp. */
} MOUEVENTINFO;

typedef MOUEVENTINFO *PMOUEVENTINFO;
```

MOUEVENTINFO Field - fs

fs (ULONG)
Mouse state.

The state of the mouse at the time of the event.

	Bit	Description
31-7		Reserved; set to zero.
6		Set if button 3 is down.
5		Set if mouse is moving and button 3 is down.
4		Set if button 2 is down.
3		Set if mouse is moving and button 2 is down.
2		Set if button 1 is down.
1		Set if mouse is moving and button 1 is down.
0		Set if mouse is moving and no buttons are down.

MOUEVENTINFO Field - row

row (LONG)
Horizontal position.

The absolute or relative row position.

MOUEVENTINFO Field - col

col (LONG)
Vertical position.

The absolute or relative column position.

MOUEVENTINFO Field - time

time (ULONG)
Timestamp.

Time stamp.

MOUQUEUEINFO

Mouse queue status structure.

```
typedef struct _MOUQUEUEINFO {  
    ULONG      cEvents; /* Current number of queue elements. */  
    ULONG      cmaxEvents; /* Maximum number of queue elements. */  
} MOUQUEUEINFO;  
  
typedef MOUQUEUEINFO *PMOUQUEUEINFO;
```

MOUQUEUEINFO Field - cEvents

cEvents (ULONG)
Current number of queue elements.

MOUQUEUEINFO Field - cmaxEvents

cmaxEvents (ULONG)
Maximum number of queue elements.

NOPTRRECT

Pointer shape collision area data structure.

```
typedef struct _NOPTRRECT {  
    ULONG      row; /* Upper-left row coordinate. */  
    ULONG      col; /* Upper-left column coordinate. */  
    ULONG      cRow; /* Lower-right row coordinate. */  
    ULONG      cCol; /* Lower-right column coordinate. */  
} NOPTRRECT;  
  
typedef NOPTRRECT *PNOPTRRECT;
```

NOPTRRECT Field - row

row (ULONG)
Upper-left row coordinate.

NOPTRRECT Field - col

col (ULONG)
Upper-left column coordinate.

NOPTRRECT Field - cRow

cRow (ULONG)
Lower-right row coordinate.

NOPTRRECT Field - cCol

cCol (ULONG)
Lower-right column coordinate.

PTRLOC

Mouse pointer position.

```
typedef struct _PTRLOC {  
    ULONG    row; /* Pointer to the row screen position. */  
    ULONG    col; /* Pointer to the column screen position. */  
} PTRLOC;
```

```
typedef PTRLOC *PPTRLOC;
```

PTRLOC Field - row

row (ULONG)
Pointer to the row screen position.

The current pointer row coordinate (pels or characters).

PTRLOC Field - col

col (ULONG)

Pointer to the column screen position.

The current pointer column coordinate (pels or characters).

PTRSHAPE

The address of a structure, in application storage, where the application stores the data necessary for the pointer device driver to return information about the Row-by-Column image for each bit plane for the mode the display is currently running. See `MouSetPtrShape` for a further description of the contents of this structure.

```
typedef struct _PTRSHAPE {
    USHORT    cb;        /* Length of the pointer buffer. */
    USHORT    col;       /* The number of columns in mouse shape. */
    USHORT    row;       /* The number of rows in mouse shape. */
    USHORT    colHot;    /* Pointer image hotspot. */
    USHORT    rowHot;    /* Pointer image hotspot. */
} PTRSHAPE;

typedef PTRSHAPE *PPTRSHAPE;
```

PTRSHAPE Field - cb

cb (USHORT)

Length of the pointer buffer.

The length of the pointer buffer available for the pointer device driver to build a row-by-column image for each bit plane for the mode the display is currently running. This value is supplied by the application. If the value is too small, pointer draw places the true length of the image in this field and returns an error.

For all OS/2 system-supported modes, `cb` is specified in bytes and is equal to:

Mono and Text Modes

For text mode, height and width must be 1, so length is always 4.

```
cb = (height in chars) * (width in chars) * 2 * 2
    = 1 * 1 * 2 * 2
    = 4
```

Graphics Mode

Width-in-pels must be a multiple of 8.

```
cb = (height in pels) * (width in pels) * (bits per pel)
```

PTRSHAPE Field - col

col (USHORT)

The number of columns in mouse shape.

In graphics modes, this field contains the pel width (columns) of the mouse shape for the session and must be greater than or equal to 1. In text modes, col must equal 1.

PTRSHAPE Field - row

row (USHORT)

The number of rows in mouse shape.

In graphics modes, this field contains the pel height (rows) of the mouse shape for the session and must be greater than or equal to 1. In text modes, row must equal 1.

PTRSHAPE Field - colHot

colHot (USHORT)

Pointer image hotspot.

This value is returned by the mouse device driver to indicate the relative column offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

PTRSHAPE Field - rowHot

rowHot (USHORT)

Pointer image hotspot.

The row of the pointer image hotspot. This value is returned by the mouse device driver to indicate the relative row offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

SCALEFACT

The address of the control block structure that contains the current row and column coordinate scaling factors. The scaling factors must be greater than or equal to 1 and less than or equal to (32K - 1).

```
typedef struct _SCALEFACT {
    ULONG    rowScale; /* Row scaling factor. */
    ULONG    colScale; /* Column coordinate scaling factor. */
} SCALEFACT;
```

```
typedef SCALEFACT *PSCALEFACT;
```

SCALEFACT Field - rowScale

rowScale (ULONG)
Row scaling factor.

SCALEFACT Field - colScale

colScale (ULONG)
Column coordinate scaling factor.

THRESHOLD

The threshold data structure.

```
typedef struct _THRESHOLD {  
    USHORT    Length;      /* Length field. */  
    USHORT    Level1;      /* First movement level. */  
    USHORT    Lev1Mult;    /* First-level multiplier. */  
    USHORT    Level2;      /* Second movement level. */  
    USHORT    Lev2Mult;    /* Second-level multiplier. */  
} THRESHOLD;  
  
typedef THRESHOLD *PTHRESHOLD;
```

THRESHOLD Field - Length

Length (USHORT)
Length field.

THRESHOLD Field - Level1

Level1 (USHORT)
First movement level.

THRESHOLD Field - Lev1Mult

Lev1Mult (USHORT)
First-level multiplier.

THRESHOLD Field - Level2

Level2 (USHORT)
Second movement level.

THRESHOLD Field - Lev2Mult

Lev2Mult (USHORT)
Second-level multiplier.

Video

This chapter describes all of the 32-bit Video functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

The chapter includes the following sections:

- [Video Functions](#)
 - [Video Data Types](#)
-

Video Functions

The video functions described in this section are:

- [VioAssociate](#)
- [VioCreateLogFont](#)
- [VioCreatePS](#)
- [VioDeleteSetId](#)
- [VioDestroyPS](#)
- [VioEndPopUp](#)
- [VioGetAnsi](#)
- [VioGetBuf](#)
- [VioGetConfig](#)
- [VioGetCp](#)
- [VioGetCurPos](#)
- [VioGetCurType](#)
- [VioGetDeviceCellSize](#)
- [VioGetMode](#)
- [VioGetOrigin](#)
- [VioGetState](#)
- [VioModeUndo](#)
- [VioModeWait](#)
- [VioPopUp](#)
- [VioQueryFonts](#)
- [VioQuerySetIds](#)
- [VioReadCharStr](#)
- [VioReadCellStr](#)
- [VioSavRedrawWait](#)
- [VioSavRedrawUndo](#)

- [VioScrollDown](#)
- [VioScrLock](#)
- [VioScrollLeft](#)
- [VioScrollRight](#)
- [VioScrollUp](#)
- [VioScrUnLock](#)
- [VioSetAnsi](#)
- [VioSetCp](#)
- [VioSetCurPos](#)
- [VioSetCurType](#)
- [VioSetDeviceCellSize](#)
- [VioSetMode](#)
- [VioSetOrigin](#)
- [VioSetState](#)
- [VioShowBuf](#)
- [VioShowPS](#)
- [VioWrtCellStr](#)
- [VioWrtCharStr](#)
- [VioWrtCharStrAtt](#)
- [VioWrtNAttr](#)
- [VioWrtNCell](#)
- [VioWrtNChar](#)
- [VioWrtTTY](#)

VioAssociate

VioAssociate - Syntax

Associate or disassociate a VIO presentation space with a device context.

```
#define INCL_VIO
#include <os2.h>

HDC      hdc;    /* Device-context handle. */
HVIO     hvps;   /* VIO presentation space handle. */
APIRET   rc;     /* Return code. */

rc = VioAssociate(hdc, hvps);
```

VioAssociate Parameter - hdc

hdc (HDC) - input
Device-context handle.

If this is NULL, a disassociation occurs.

VioAssociate Parameter - hvps

hvps (HVIO) - input
VIO presentation space handle.

This is returned from [VioCreatePS](#).

VioAssociate Return Value - rc

rc (APIRET) - returns
Return code.

VioAssociate returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
495	ERROR_VIO_NOT_PRESEN_MGR_SG
499	ERROR_VIO_ASSOCIATED_DC

VioAssociate - Parameters

hdc (HDC) - input
Device-context handle.

If this is NULL, a disassociation occurs.

hvps (HVIO) - input
VIO presentation space handle.

This is returned from [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioAssociate returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
495	ERROR_VIO_NOT_PRESEN_MGR_SG
499	ERROR_VIO_ASSOCIATED_DC

VioAssociate - Remarks

Subsequent VIO calls to this VIO presentation space will direct output to the specified device context.

If a null handle is specified for the device context, the presentation space is disassociated from any device context.

An associated presentation space or device context cannot be associated.

The screen device context is the only kind of device that can be associated with a VIO presentation space.

VioAssociate - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioCreateLogFont

VioCreateLogFont - Syntax

Specify a font for use by a VIO session.

```
#define INCL_VIO
#include <os2.h>

PFATTRS    attrs; /* Pointer to the font attribute structure. */
ULONG      lcid;  /* Local identifier. */
STR8       Name;  /* Logical font name. */
HVIO       hvps;  /* VIO presentation-space handle. */
APIRET     rc;    /* Return code. */

rc = VioCreateLogFont(attrs, lcid, Name, hvps);
```

VioCreateLogFont Parameter - attrs

attrs ([PFATTRS](#)) - input
Pointer to the font attribute structure.

VioCreateLogFont Parameter - Lcid

lcid (ULONG) - input
Local identifier.

This must be a value in the range 1 to 3. It is an error if *lcid* is already in use.

VioCreateLogFont Parameter - Name

Name (STR8) - input
Logical font name.

This string is optional and is used to describe the logical font.

VioCreateLogFont Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

VioCreateLogFont Return Value - rc

rc (APIRET) - returns
Return code.

VioCreateLogFont returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioCreateLogFont - Parameters

attrs ([PFATTRS](#)) - input
Pointer to the font attribute structure.

lcid (ULONG) - input
Local identifier.

This must be a value in the range 1 to 3. It is an error if *lcid* is already in use.

Name (STR8) - input
Logical font name.

This string is optional and is used to describe the logical font.

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioCreateLogFont returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioCreateLogFont - Remarks

The system selects the font most closely matching the specified font from the set of monospaced fonts installed in the system.

In OS/2 2.x, *hvp*s cannot be 0.

VioCreateLogFont - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioCreatePS

VioCreatePS - Syntax

Create a VIO presentation space.

```
#define INCL_VIO
#include <os2.h>

PHVIO    phvps;    /* Pointer to the presentation-space handle. */
ULONG    Rows;     /* Number of rows. */
ULONG    Columns;  /* Number of columns. */
ULONG    Format;    /* Format of the attributes */
ULONG    AttrBytes; /* Number of attribute bytes. */
HVIO     hvps;     /* Reserved. Must be 0. */
APIRET   rc;       /* Return code. */

rc = VioCreatePS(phvps, Rows, Columns, Format,
                AttrBytes, hvps);
```

VioCreatePS Parameter - phvps

phvps (PHVIO) - output
Pointer to the presentation-space handle.

The location where the newly created presentation-space handle is to be returned.

VioCreatePS Parameter - Rows

Rows (ULONG) - input
Number of rows.

The number of rows in the presentation space. The maximum value allowed is 255.

VioCreatePS Parameter - Columns

Columns (ULONG) - input
Number of columns.

The number of columns in the presentation space. The maximum value allowed is 255.

VioCreatePS Parameter - Format

Format (ULONG) - input
Format of the attributes

The attributes may be one of the following format types:

- | | |
|---|----------------|
| 1 | VGA compatible |
| 2 | Unicode |
| 3 | MFI compatible |
-

VioCreatePS Parameter - AttrBytes

AttrBytes (ULONG) - input
Number of attribute bytes.

This is used along with the format to select the attribute structure. This field has a value of 1, 2, or 3.

VioCreatePS Parameter - hvps

hvps (HVIO) - input
Reserved. Must be 0.

VioCreatePS Return Value - rc

rc (APIRET) - returns
Return code.

VioCreatePS returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioCreatePS - Parameters

phvps (PHVIO) - output
Pointer to the presentation-space handle.

The location where the newly created presentation-space handle is to be returned.

Rows (ULONG) - input
Number of rows.

The number of rows in the presentation space. The maximum value allowed is 255.

Columns (ULONG) - input
Number of columns.

The number of columns in the presentation space. The maximum value allowed is 255.

Format (ULONG) - input
Format of the attributes

The attributes may be one of the following format types:

1	VGA compatible
2	Unicode
3	MFI compatible

AttrBytes (ULONG) - input
Number of attribute bytes.

This is used along with the format to select the attribute structure. This field has a value of 1, 2, or 3.

hvps (HVIO) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioCreatePS returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioCreatePS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

VioDeleteSetId

VioDeleteSetId - Syntax

Make the logical font no longer available via the local ID.

```
#define INCL_VIO
#include <os2.h>

ULONG      lcid; /* Local ID for the font. */
HVIO       hvps; /* VIO presentation space handle. */
APIRET     rc;   /* Return code. */

rc = VioDeleteSetId(lcid, hvps);
```

VioDeleteSetId Parameter - lcid

lcid (ULONG) - input
Local ID for the font.

VioDeleteSetId Parameter - hvps

hvps (HVIO) - input
VIO presentation space handle.

This is either 0 to indicate the default VIO session, or a value returned by VioCreatePS.

VioDeleteSetId Return Value - rc

rc (APIRET) - returns
Return code.

VioDeleteSetId returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioDeleteSetId - Parameters

lcid (ULONG) - input
Local ID for the font.

hvps (HVIO) - input
VIO presentation space handle.

This is either 0 to indicate the default VIO session, or a value returned by VioCreatePS.

rc (APIRET) - returns
Return code.

VioDeleteSetId returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioDeleteSetId - Remarks

After this call, the *lcid* is available for reuse.

In OS/2 2.x, *hvps* cannot be 0.

VioDeleteSetId - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioDestroyPS

VioDestroyPS - Syntax

Destroy the VIO presentation space.

```
#define INCL_VIO
#include <os2.h>
```

```
HVIO      hvps; /* VIO presentation-space handle. */
APIRET    rc;   /* Return code. */

rc = VioDestroyPS(hvps);
```

VioDestroyPS Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is a value returned by [VioCreatePS](#).

VioDestroyPS Return Value - rc

rc (APIRET) - returns
Return code.

VioDestroyPS returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
499	ERROR_VIO_ASSOCIATED_DC

VioDestroyPS - Parameters

hvps (HVIO) - input
VIO presentation-space handle.

This is a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioDestroyPS returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
499	ERROR_VIO_ASSOCIATED_DC

VioDestroyPS - Remarks

The presentation space must not be associated with a device context when VioDestroyPS is called.

The VIO presentation-space handle is invalid after this call.

After this call, the local ID for the font is available for reuse.

VioDestroyPS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioEndPopUp

VioEndPopUp - Syntax

VioEndPopUp redirects video output back to the normal video buffer. This function should be issued by the application when it no longer requires the temporary screen obtained through a previous [VioPopUp](#) call.

```
#define INCL_VIO
#include <os2.h>

HVIO      VioHandle; /* Reserved. Must be 0. */
APIRET    rc;        /* Return code. */

rc = VioEndPopUp(VioHandle);
```

VioEndPopUp Parameter - VioHandle

VioHandle (HVIO) - input
Reserved. Must be 0.

VioEndPopUp Return Value - rc

rc (APIRET) - returns
Return code.

VioEndPopUp returns one of the following values:

0	NO_ERROR
405	ERROR_VIO_NO_POPUP
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioEndPopUp - Parameters

VioHandle (HVIO) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioEndPopUp returns one of the following values:

0	NO_ERROR
405	ERROR_VIO_NO_POPUP
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioEndPopUp - Remarks

When the application issues a VioEndPopUp call, all video calls are directed to the application's normal video buffer. An error is returned if the call is issued with a non-zero handle.

VioEndPopUp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetAnsi

VioGetAnsi - Syntax

VioGetAnsi returns the current ANSI status On/Off state.

```
#define INCL_VIO
#include <os2.h>
```

```
PULONG    Indicator; /* Address of the current ANSI status. */
HVIO      VioHandle; /* Presentation-space handle. */
APIRET    rc;        /* Return code. */
```

```
rc = VioGetAnsi(Indicator, VioHandle);
```

VioGetAnsi Parameter - Indicator

Indicator (PULONG) - output

Address of the current ANSI status.

The ANSI status may be one of the following values:

1	ANSI is active
0	ANSI is not active

VioGetAnsi Parameter - VioHandle

VioHandle (HVIO) - input

Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioGetAnsi Return Value - rc

rc (APIRET) - returns

Return code.

VioGetAnsi returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetAnsi - Parameters

Indicator (PULONG) - output

Address of the current ANSI status.

The ANSI status may be one of the following values:

1	ANSI is active
0	ANSI is not active

VioHandle (HVIO) - input

Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns

Return code.

VioGetAnsi returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetAnsi - Remarks

When the application issues a [VioEndPopUp](#) call, all video calls are directed to the application's normal video buffer. An error is returned if the call is issued with a non-zero handle.

VioGetAnsi - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetBuf

VioGetBuf - Syntax

VioGetBuf returns the address of the logical video buffer (LVB).

```
#define INCL_VIO
#include <os2.h>

PULONG    LVBPtr;    /* Pointer to the logical video buffer address. */
PULONG    Length;    /* Pointer to the length of the buffer, in bytes. */
HVIO      VioHandle; /* Presentation-space handle. */
APIRET    rc;        /* Return code. */

rc = VioGetBuf(LVBPtr, Length, VioHandle);
```

VioGetBuf Parameter - LVBPtr

LVBPtr (PULONG) - output
Pointer to the logical video buffer address.

VioGetBuf Parameter - Length

Length (PULONG) - output

Pointer to the length of the buffer, in bytes.

The length is the number of rows, times the number of columns, times the size of the cell.

VioGetBuf Parameter - VioHandle

VioHandle (HVIO) - input

Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioGetBuf Return Value - rc

rc (APIRET) - returns

Return code.

VioGetBuf returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioGetBuf - Parameters

LVBPtr (PULONG) - output

Pointer to the logical video buffer address.

Length (PULONG) - output

Pointer to the length of the buffer, in bytes.

The length is the number of rows, times the number of columns, times the size of the cell.

VioHandle (HVIO) - input

Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns

Return code.

VioGetBuf returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioGetBuf - Remarks

An application using VioGetBuf can prepare a screen in the application's own logical video buffer (LVB) offline. When the application is in the foreground, the physical screen buffer is updated from the LVB when VioShowBuf is issued. When the application runs in the background, the physical screen buffer is updated when the application is switched to the foreground.

Once VioGetBuf is issued, all VioWrtXX calls issued while the application is running in the foreground are written to the physical display buffer and LVB.

VioGetMode can be used to determine the dimensions of the buffer.

If [VioSetMode](#) is issued following a VioGetBuf call, the size of the LVB is changed, and VioGetBuf must be reissued.

VioGetBuf - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetConfig

VioGetConfig - Syntax

VioGetConfig returns the video display configuration.

```
#define INCL_VIO
#include <os2.h>

ULONG          ConfigID;      /* Configuration ID. */
PVIOCONFIGINFO ConfigData;    /* Pointer to the configuration data. */
HVIO           VioHandle;     /* Presentation-space handle. */
APIRET         rc;            /* Return code. */

rc = VioGetConfig(ConfigID, ConfigData, VioHandle);
```

VioGetConfig Parameter - ConfigID

ConfigID (ULONG) - input
Configuration ID.

Identifies the display configuration for which information is being requested:

	Value	Definition
0		Current configuration
1		Primary configuration
2		Secondary configuration

VioGetConfig Parameter - ConfigData

ConfigData ([PVIOCONFIGINFO](#)) - output
Pointer to the configuration data.

VioGetConfig Parameter - VioHandle

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioGetConfig Return Value - rc

rc (APIRET) - returns
Return code.

VioGetConfig returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioGetConfig - Parameters

ConfigID (ULONG) - input
Configuration ID.

Identifies the display configuration for which information is being requested:

	Value	Definition
0		Current configuration
1		Primary configuration

ConfigData ([PVIOCONFIGINFO](#)) - output
Pointer to the configuration data.

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioGetConfig returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioGetConfig - Remarks

The values returned might not be correct if the adapter cannot be properly identified or if the device is not capable of returning its settings.

VioGetConfig - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetCp

VioGetCp - Syntax

VioGetCp queries the code page currently used to display text data.

```
#define INCL_VIO
#include <os2.h>

ULONG      Reserved;    /* Reserved. Must be 0. */
PUSHORT    CodePageID;  /* Code-page ID. */
HVIO       VioHandle;   /* Presentation-space handle. */
APIRET     rc;          /* Return code. */

rc = VioGetCp(Reserved, CodePageID, VioHandle);
```

VioGetCp Parameter - Reserved

Reserved (ULONG) - input
Reserved. Must be 0.

VioGetCp Parameter - CodePageID

CodePageID (USHORT) - output
Code-page ID.

The address of a word in the application's data area. The current video code page is returned in this word.

VioGetCp Parameter - VioHandle

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioGetCp Return Value - rc

rc (APIRET) - returns
Return code.

VioGetCp returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetCp - Parameters

Reserved (ULONG) - input
Reserved. Must be 0.

CodePageID (USHORT) - output
Code-page ID.

The address of a word in the application's data area. The current video code page is returned in this word.

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioGetCp returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetCp - Remarks

The display code-page ID, previously set by [VioSetCp](#) or inherited from the requesting process, is returned to the caller. The returned code-page tag is the currently active code page.

VioGetCp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetCurPos

VioGetCurPos - Syntax

VioGetCurPos returns the coordinates of the cursor.

```
#define INCL_VIO
#include <os2.h>

PULONG      Row;          /* Row return data. */
PUSHORT     Column;       /* Column return data. */
HVIO        VioHandle;    /* Presentation-space handle. */
APIRET      rc;           /* Return code. */

rc = VioGetCurPos(Row, Column, VioHandle);
```

VioGetCurPos Parameter - Row

Row (PULONG) - output
Row return data.

Address of the current row position of the cursor, where 0 is the top row.

VioGetCurPos Parameter - Column

Column (PUSHORT) - output
Column return data.

The address of the current column position of the cursor, where 0 is the leftmost column.

VioGetCurPos Parameter - VioHandle

VioHandle (HVIO) - input
Presentation-space handle.

This must be zero, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#)

VioGetCurPos Return Value - rc

rc (APIRET) - returns
Return code.

VioGetCurPos returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetCurPos - Parameters

Row (PULONG) - output
Row return data.

Address of the current row position of the cursor, where 0 is the top row.

Column (PUSHORT) - output
Column return data.

The address of the current column position of the cursor, where 0 is the leftmost column.

VioHandle (HVIO) - input

Presentation-space handle.

This must be zero, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#)

rc (APIRET) - returns
Return code.

VioGetCurPos returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetCurPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

VioGetCurType

VioGetCurType - Syntax

VioGetCurType returns the cursor type.

```
#define INCL_VIO
#include <os2.h>

PVIOCURSORINFO    CursorData; /* Cursor characteristics. */
HVIO              VioHandle;  /* Presentation-space handle. */
APIRET            rc;         /* Return code. */

rc = VioGetCurType(CursorData, VioHandle);
```

VioGetCurType Parameter - CursorData

CursorData ([PVIOCURSORINFO](#)) - output
Cursor characteristics.

Address of the cursor-characteristics structure.

VioGetCurType Parameter - VioHandle

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#)

VioGetCurType Return Value - rc

rc (APIRET) - returns
Return code.

VioGetCurType returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetCurType - Parameters

CursorData ([PVIOCURSORINFO](#)) - output
Cursor characteristics.

Address of the cursor-characteristics structure.

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#)

rc (APIRET) - returns
Return code.

VioGetCurType returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetCurType - Remarks

The cursor start line (*yStart*) and cursor end line (*cEnd*) were originally specified as percentages on [VioSetCurType](#) (using negative values); the positive values, into which they were translated, are returned. See [VioSetCurType](#) for more information on how percentages can be used to set *yStart* and *cEnd* independent of the number of scan lines per character cell.

VioGetCurType - Topics

Select an item:

VioGetDeviceCellSize

VioGetDeviceCellSize - Syntax

VioGetDeviceCellSize returns the size of the current character cell in pels.

```
#define INCL_VIO
#include <os2.h>

PULONG    Height;
PULONG    Width;
HVIO      hvps;    /* VIO presentation-space handle. */
APIRET    rc;      /* Return code. */

rc = VioGetDeviceCellSize(Height, Width, hvps);
```

VioGetDeviceCellSize Parameter - Height

Height (PULONG) - input
Pointer to the height of the character cell in pels.

VioGetDeviceCellSize Parameter - Width

Width (PULONG) - input
Pointer to the width of the character cell in pels.

VioGetDeviceCellSize Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is a value returned by [VioCreatePS](#).

VioGetDeviceCellSize Return Value - rc

rc (APIRET) - returns
Return code.

VioGetDeviceCellSize returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetDeviceCellSize - Parameters

Height (PULONG) - input
Pointer to the height of the character cell in pels.

Width (PULONG) - input
Pointer to the width of the character cell in pels.

hvps (HVIO) - input
VIO presentation-space handle.

This is a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioGetDeviceCellSize returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetDeviceCellSize - Remarks

In OS/2 2.x, *hvps* cannot be 0.

VioGetDeviceCellSize - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetMode

VioGetMode - Syntax

VioGetMode returns the mode of the display.

```
#define INCL_VIO
#include <os2.h>

PVIOMODEINFO    ModeData;    /* Mode characteristics. */
HVIO            VioHandle;    /* VIO presentation-space handle. */
APIRET          rc;          /* Return code. */

rc = VioGetMode(ModeData, VioHandle);
```

VioGetMode Parameter - ModeData

ModeData ([PVIOMODEINFO](#)) - in/out
Mode characteristics.

The address of a structure where mode characteristics are returned.

VioGetMode Parameter - VioHandle

VioHandle ([HVIO](#)) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioGetMode Return Value - rc

rc ([APIRET](#)) - returns
Return code.

VioGetMode returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioGetMode - Parameters

ModeData ([PVIOMODEINFO](#)) - in/out
Mode characteristics.

The address of a structure where mode characteristics are returned.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioGetMode returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioGetMode - Remarks

See [VioSetMode](#) for examples.

VioGetMode - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetOrigin

VioGetOrigin - Syntax

Get the position at which the presentation space maps to the window.

```
#define INCL_VIO
#include <os2.h>

PULONG      Row;
PULONG      Column;
HVIO        hvps; /* VIO presentation-space handle. */
APIRET      rc;   /* Return code. */

rc = VioGetOrigin(Row, Column, hvps);
```

VioGetOrigin Parameter - Row

Row (PULONG) - input
Location to return the top-most row shown in the window.

VioGetOrigin Parameter - Column

Column (PULONG) - input
Location to return the left-most column shown in the window.

VioGetOrigin Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

VioGetOrigin Return Value - rc

rc (APIRET) - returns
Return code.

VioGetOrigin returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetOrigin - Parameters

Row (PULONG) - input
Location to return the top-most row shown in the window.

Column (PULONG) - input
Location to return the left-most column shown in the window.

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioGetOrigin returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioGetOrigin - Remarks

In OS/2 2.x, *hvps* cannot be 0.

VioGetOrigin - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioGetState

VioGetState - Syntax

VioGetState returns the current settings of the palette registers, overscan (border) color, blink/background intensity switch, color registers, underline location, or target VioSetMode display configuration.

```
#define INCL_VIO
#include <os2.h>

PVOID    RequestBlock; /* Request block. */
HVIO     VioHandle;    /* Reserved. Must be 0. */
APIRET   rc;           /* Return code. */

rc = VioGetState(RequestBlock, VioHandle);
```

VioGetState Parameter - RequestBlock

RequestBlock (PVIOD) - in/out
Request block.

The address of the video-state structures consisting of six different structures depending on the request type:

Type	Definition
0	Get palette registers
1	Get overscan (border) color
2	Get blink/background intensity switch
3	Get color registers
4	Reserved
5	Get the scan line for underlining
6	Get target VioSetMode display configuration

The six structures, depending on request type, are:

- [VIOPALSTATE](#)
- [VIOOVERSCAN](#)
- [VIOINTENSITY](#)
- [VIOCOLORREG](#)
- [VIOSETLINELOC](#)
- [VIOSETTARGET](#)

VioGetState Parameter - VioHandle

VioHandle (HVIO) - input
Reserved. Must be 0.

VioGetState Return Value - rc

rc (APIRET) - returns
Return code.

VioGetState returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioGetState - Parameters

RequestBlock (PVIOD) - in/out
Request block.

The address of the video-state structures consisting of six different structures depending on the request type:

Type	Definition
0	Get palette registers
1	Get overscan (border) color
2	Get blink/background intensity switch
3	Get color registers
4	Reserved
5	Get the scan line for underlining

The six structures, depending on request type, are:

- [VIOPALSTATE](#)
- [VIOOVERSCAN](#)
- [VIOINTENSITY](#)
- [VIOCOLORREG](#)
- [VIOSETULINELOC](#)
- [VIOSETTARGET](#)

VioHandle (HVIO) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioGetState returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioGetState - Remarks

Note: VioGetState allows access to hardware-dependent features. Not all video hardware will honor these settings or return valid settings.

VioGetState - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioModeUndo

VioModeUndo - Syntax

VioModeUndo allows one thread within a process to cancel a [VioModeWait](#) issued by another thread within the same process.

```
#define INCL_VIO
#include <os2.h>

ULONG      OwnerIndic; /* Ownership indicator. */
```

```

ULONG      KillIndic;    /* Terminate indicator */
ULONG      Reserved;     /* Reserved. Must be 0. */
APIRET     rc;           /* Return code. */

```

```
rc = VioModeUndo(OwnerIndic, KillIndic, Reserved);
```

VioModeUndo Parameter - OwnerIndic

OwnerIndic (ULONG) - input
Ownership indicator.

Indicates whether the thread issuing VioModeUndo wants ownership of [VioModeWait](#) to be reserved for its process.

Value	Definition
0	Reserve ownership
1	Give up ownership.

VioModeUndo Parameter - KillIndic

KillIndic (ULONG) - input
Terminate indicator

Indicates whether the thread (with the outstanding [VioModeWait](#)) should return an error code or be terminated.

Value	Definition
0	Return error code.
1	Terminate thread.

VioModeUndo Parameter - Reserved

Reserved (ULONG) - input
Reserved. Must be 0.

VioModeUndo Return Value - rc

rc (APIRET) - returns
Return code.

VioModeUndo returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
427	ERROR_VIO_NO_MODE_THREAD
430	ERROR_VIO_ILLEGAL_DURING_POPUP
486	ERROR_VIO_BAD_RESERVE

VioModeUndo - Parameters

OwnerIndic (ULONG) - input
Ownership indicator.

Indicates whether the thread issuing VioModeUndo wants ownership of [VioModeWait](#) to be reserved for its process.

Value	Definition
0	Reserve ownership
1	Give up ownership.

KillIndic (ULONG) - input
Terminate indicator

Indicates whether the thread (with the outstanding [VioModeWait](#)) should return an error code or be terminated.

Value	Definition
0	Return error code.
1	Terminate thread.

Reserved (ULONG) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioModeUndo returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
427	ERROR_VIO_NO_MODE_THREAD
430	ERROR_VIO_ILLEGAL_DURING_POPUP
486	ERROR_VIO_BAD_RESERVE

VioModeUndo - Remarks

VioModeUndo can be issued only by a thread within the process that owns [VioModeWait](#). The thread issuing VioModeUndo can either reserve ownership of the [VioModeWait](#) call for its process or give up ownership. The thread whose [VioModeWait](#) is canceled is optionally terminated.

VioModeUndo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioModeWait

VioModeWait - Syntax

VioModeWait notifies a graphics-mode application when it must restore its video mode, state, and modified display-adapter registers. The return from this call provides the notification.

```
#define INCL_VIO
#include <os2.h>

ULONG      RequestType; /* Request type. */
PULONG     NotifyType;  /* Notify type. */
ULONG      Reserved;    /* Reserved. Must be 0. */
APIRET     rc;           /* Return code. */

rc = VioModeWait(RequestType, NotifyType,
                  Reserved);
```

VioModeWait Parameter - RequestType

RequestType (ULONG) - input
Request type.

Valid request types are:

0 Indicates the application wants to be notified at the end of a pop-up to restore its mode.

VioModeWait Parameter - NotifyType

NotifyType (PULONG) - output
Notify type.

Address of the of the operation to be performed by the application returning from VioModeWait. Valid notify types are:

0 Restore mode.

VioModeWait Parameter - Reserved

Reserved (ULONG) - input
Reserved. Must be 0.

VioModeWait Return Value - rc

rc (APIRET) - returns
Return code.

VioModeWait returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
423	ERROR_VIO_RETURN
424	ERROR_SCS_INVALID_FUNCTION
428	ERROR_VIO_NO_SAVE_RESTORE_THD
430	ERROR_VIO_ILLEGAL_DURING_POPUP

VioModeWait - Parameters

RequestType (ULONG) - input
Request type.

Valid request types are:

0	Indicates the application wants to be notified at the end of a pop-up to restore its mode.
---	--

NotifyType (PULONG) - output
Notify type.

Address of the of the operation to be performed by the application returning from VioModeWait. Valid notify types are:

0	Restore mode.
---	---------------

Reserved (ULONG) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioModeWait returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
423	ERROR_VIO_RETURN
424	ERROR_SCS_INVALID_FUNCTION
428	ERROR_VIO_NO_SAVE_RESTORE_THD
430	ERROR_VIO_ILLEGAL_DURING_POPUP

VioModeWait - Remarks

At the completion of an application or hard-error pop-up (see [VioPopUp](#)), the OS/2 operating system notifies the session that was originally interrupted for the pop-up to restore its mode. The return from VioModeWait provides that notification. The thread that issued the call must perform the restore and then immediately reissue VioModeWait.

When an application's VioModeWait thread is notified, the thread must restore its video mode, state, and modified display-adaptor registers. An application's VioModeWait thread does not restore the physical display buffer. The OS/2 operating system saves and/or restores the physical display buffer over a pop-up.

Only one process for a session can issue VioModeWait. The first process that issues VioModeWait becomes the owner of this function. (See [VioModeUndo](#).)

An application must issue VioModeWait only if it writes directly to the registers on the display adapter. Otherwise, the application can allow the OS/2 operating system to perform the required restoration by not issuing VioModeWait.

When an application issues VioModeWait, it is also required to issue [VioSavRedrawWait](#) to be notified at screen switch time to perform a full

save or restoration (see [VioSavRedrawWait](#)). Two application threads must be dedicated to performing these operations.

VioModeWait - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioPopUp

VioPopUp - Syntax

VioPopUp displays a temporary screen with a message to the user.

```
#define INCL_VIO
#include <os2.h>

PULONG    Options;    /* Option flags. */
HVIO      VioHandle;  /* Presentation-space handle. */
APIRET    rc;         /* Return code. */

rc = VioPopUp(Options, VioHandle);
```

VioPopUp Parameter - Options

Options (PULONG) - input
Option flags.

Bit	Description
31-2	Reserved. Set to 0.
1	0 = Non-transparent operation. The video mode is set to a text mode. The screen is cleared, and the cursor is positioned at the upper-left corner of the screen. 1 = Transparent operation. If the video mode of the outgoing foreground session is a text mode, no mode change occurs. The screen is not cleared, and the cursor remains at its current position. If the video mode is not text, or if this session has a VioSavRedrawWait active, the pop-up is refused. The OS/2 operating system is responsible for saving and restoring the physical display buffer of the previous foreground session around a pop-up. This is true whether transparent or nontransparent operation is selected.
0	0 = Return with unique error code, ERROR_VIO_EXISTING_POPUP, if pop-up is not immediately available.

1 = Wait if pop-up is not immediately available.

VioPopUp Parameter - VioHandle

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioPopUp Return Value - rc

rc (APIRET) - returns
Return code.

VioPopUp returns one of the following values:

0	NO_ERROR
405	ERROR_VIO_NO_POPUP
406	ERROR_VIO_EXISTING_POPUP
421	ERROR_VIO_INVALID_PARMs
483	ERROR_VIO_TRANSPARENT_POPUP

VioPopUp - Parameters

Options (PULONG) - input
Option flags.

Bit	Description
31-2	Reserved. Set to 0.
1	<p>0 = Non-transparent operation. The video mode is set to a text mode. The screen is cleared, and the cursor is positioned at the upper-left corner of the screen.</p> <p>1 = Transparent operation. If the video mode of the outgoing foreground session is a text mode, no mode change occurs. The screen is not cleared, and the cursor remains at its current position. If the video mode is not text, or if this session has a VioSavRedrawWait active, the pop-up is refused.</p> <p>The OS/2 operating system is responsible for saving and restoring the physical display buffer of the previous foreground session around a pop-up. This is true whether transparent or nontransparent operation is selected.</p>
0	<p>0 = Return with unique error code, ERROR_VIO_EXISTING_POPUP, if pop-up is not immediately available.</p> <p>1 = Wait if pop-up is not immediately available.</p>

VioHandle (HVIO) - input
Presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns

Return code.

VioPopUp returns one of the following values:

0	NO_ERROR
405	ERROR_VIO_NO_POPUP
406	ERROR_VIO_EXISTING_POPUP
421	ERROR_VIO_INVALID_PARMS
483	ERROR_VIO_TRANSPARENT_POPUP

VioPopUp - Remarks

VioPopUp is normally issued by the application when it is running in the background and wishes to immediately display a message to the user without waiting to become the active foreground session.

When an application process issues VioPopUp, it should wait for the return from the request. If the process allows any of its threads to write to the screen before VioPopUp returns a successful return code, the screen output might be directed to the application's normal video buffer rather than to the pop-up screen. If the process allows any of its threads to issue keyboard or mouse calls before VioPopUp returns a successful return code, the input is directed from the application's normal session. Once the process that issued VioPopUp receives a successful return code, video and keyboard calls issued by any of the threads in the pop-up process are directed to the pop-up screen. This continues until the process issues [VioEndPopUp](#). At that time, video and keyboard calls resume being directed to the application's normal video buffer.

Only one pop-up can exist at any time. If a process requests a pop-up and a pop-up already exists, the process can either wait for the prior pop-up to end or receive an immediate return with an error code. The error code indicates that the operation failed due to an existing pop-up having captured the screen.

Video pop-ups provide a mechanism for a background application to notify the operator of an abnormal event that requires the operator to take some action. When considering the suitability of using pop-ups in a particular situation, the possible disruptive effect of pop-ups to the operator should be considered. If the operator were interrupted frequently by pop-ups issued by background applications, the operator would not work effectively with the foreground application.

While a video pop-up is in the foreground, the operator cannot use the hot key to switch to another application or to the shell. Before the operator can switch to another application, or switch the shell to the foreground, the pop-up application must issue [VioEndPopUp](#).

While a video pop-up is in effect, all video calls from the previous foreground session are blocked until the process that issued VioPopUp issues [VioEndPopUp](#).

When VioPopUp is issued, only the process within the session that issued VioPopUp is brought to the foreground. Assuming the session was already the foreground session, any video calls issued by other processes in that session are blocked until the process that issued VioPopUp issues [VioEndPopUp](#). VioEndPopUp.

DosExecPgm cannot be issued by a process during a pop-up. The following video calls are the only calls that can be issued during the pop-up by the process that issued VioPopUp:

VioEndPopUp	VioScrollLeft
VioGetConfig	VioSetCurPos
VioGetCp	VioSetCurType
VioGetAnsi	VioSetCp
VioGetState	VioSetState
VioGetCurPos	VioWrtNChar
VioGetCurType	VioWrtNAttr
VioGetMode	VioWrtNCell
VioReadCellStr	VioWrtCharStr
VioReadCharStr	VioWrtCharStrAtt
VioScrollRight	VioWrtCellStr
VioScrollUp	VioWrtTTY
VioScrollDown	

This function can be used from within a PM application. Kbdxxx, Mouxxx, and Vioxxx calls with a zero handle are all allowed between VioPopUp and [VioEndPopUp](#) and are directed to the pop-up screen.

VioPopUp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioQueryFonts

VioQueryFonts - Syntax

VioQueryFonts gets the position at which the presentation space maps to the window.

```
#define INCL_VIO
#include <os2.h>

PULONG      Remfonts; /* Number of fonts not returned. */
PFONTMETRICS Metrics; /* Font metrics. */
ULONG      Metlen; /* Maximum length of data. */
PULONG      Fonts; /* Number of fonts. */
PSZ         Facename; /* Face name of fonts. */
ULONG      Options; /* Controls which fonts are selected. */
HVIO        hvps; /* VIO presentation-space handle. */
APIRET      rc; /* Return code. */

rc = VioQueryFonts(Remfonts, Metrics, Metlen,
                  Fonts, Facename, Options, hvps);
```

VioQueryFonts Parameter - Remfonts

Remfonts (PULONG) - output
Number of fonts not returned.

The number of fonts for which information is not returned.

VioQueryFonts Parameter - Metrics

Metrics (PFONTMETRICS) - output
Font metrics.

Matching font metrics are returned in this buffer.

VioQueryFonts Parameter - Metlen

Metlen (ULONG) - input
Maximum length of data.

The maximum length of data to be returned for each record.

VioQueryFonts Parameter - Fonts

Fonts (PULONG) - in/out
Number of fonts.

The location of the number of fonts to be returned on input. This is updated with the actual number of fonts returned on output.

VioQueryFonts Parameter - Facename

Facename (PSZ) - input
Face name of fonts.

The face name of fonts desired, or NULL to indicate that all applicable fonts should be returned.

VioQueryFonts Parameter - Options

Options (ULONG) - input
Controls which fonts are selected.

Valid values are:

VQF_PUBLIC
VQF_PRIVATE
VQF_ALL

Return only public fonts.
Return only private fonts.
Return both public and private fonts.

VioQueryFonts Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

VioQueryFonts Return Value - rc

rc (APIRET) - returns
Return code.

VioQueryFonts returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioQueryFonts - Parameters

Remfonts (PULONG) - output
Number of fonts not returned.

The number of fonts for which information is not returned.

Metrics ([PFONTMETRICS](#)) - output
Font metrics.

Matching font metrics are returned in this buffer.

Metlen (ULONG) - input
Maximum length of data.

The maximum length of data to be returned for each record.

Fonts (PULONG) - in/out
Number of fonts.

The location of the number of fonts to be returned on input. This is updated with the actual number of fonts returned on output.

Facename (PSZ) - input
Face name of fonts.

The face name of fonts desired, or NULL to indicate that all applicable fonts should be returned.

Options (ULONG) - input
Controls which fonts are selected.

Valid values are:

VQF_PUBLIC	Return only public fonts.
VQF_PRIVATE	Return only private fonts.
VQF_ALL	Return both public and private fonts.

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioQueryFonts returns one of the following values:

0
421
436

NO_ERROR
ERROR_VIO_INVALID_PARMS
ERROR_VIO_INVALID_HANDLE

VioQueryFonts - Remarks

By inspecting the returned font metrics, the application can choose the font that best meets its requirements.

All metrics are returned in pel coordinates.

In OS/2 2.x, *hvp*s cannot be zero.

VioQueryFonts - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioQuerySetIds

VioQuerySetIds - Syntax

VioQuerySetIds returns information about VIO local identifiers.

```
#define INCL_VIO
#include <os2.h>

PULONG    lcids; /* Array of local identifiers. */
PSTR8     Names;
PULONG    Types;
PULONG    count; /* Number of objects to be queried. */
HVIO      hvps;  /* VIO presentation-space handle. */
APIRET    rc;    /* Return code. */

rc = VioQuerySetIds(lcids, Names, Types, count,
                   hvps);
```

VioQuerySetIds Parameter - lcids

lcids (PULONG) - output
Array of local identifiers.

VioQuerySetIds Parameter - Names

Names (PSTR8) - output
An array of 8 character names associated with the *lcids* .

VioQuerySetIds Parameter - Types

Types (PULONG) - output
An array of types associated with each *lcids* .

VioQuerySetIds Parameter - count

count (PULONG) - input
Number of objects to be queried.

The maximum value in use is 3.

VioQuerySetIds Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

VioQuerySetIds Return Value - rc

rc (APIRET) - returns
Return code.

VioQuerySetIds returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioQuerySetIds - Parameters

Ids (PULONG) - output
Array of local identifiers.

Names (PSTR8) - output
An array of 8 character names associated with the *Ids* .

Types (PULONG) - output
An array of types associated with each *Ids* .

count (PULONG) - input
Number of objects to be queried.

The maximum value in use is 3.

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session, or a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioQuerySetIds returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioQuerySetIds - Remarks

In OS/2 2.x, *hvps* cannot be 0.

VioQuerySetIds - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioReadCharStr

VioReadCharStr - Syntax

VioReadCharStr reads a string of characters from the display, starting at the specified location.

```
#define INCL_VIO
#include <os2.h>

PCH      CharStr;    /* Character buffer. */
PULONG   Length;     /* Length of buffer. */
ULONG    Row;        /* Starting row location. */
ULONG    Column;     /* Starting column location. */
HVIO     VioHandle;  /* Vio presentation-space handle. */
APIRET   rc;         /* Return code. */

rc = VioReadCharStr(CharStr, Length, Row,
                    Column, VioHandle);
```

VioReadCharStr Parameter - CharStr

CharStr (PCH) - output
Character buffer.

Address of the buffer where the character string is returned.

VioReadCharStr Parameter - Length

Length (PULONG) - in/out
Length of buffer.

Address of the buffer length in bytes.

VioReadCharStr Parameter - Row

Row (ULONG) - input
Starting row location.

Starting row of the field to read, where 0 is the top row.

VioReadCharStr Parameter - Column

Column (ULONG) - input
Starting column location.

Starting column of the field to read, where 0 is the leftmost column.

VioReadCharStr Parameter - VioHandle

VioHandle (HVIO) - input
Vio presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioReadCharStr Return Value - rc

rc (APIRET) - returns
Return code.

VioReadCharStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioReadCharStr - Parameters

CharStr (PCH) - output
Character buffer.

Address of the buffer where the character string is returned.

Length (PULONG) - in/out
Length of buffer.

Address of the buffer length in bytes.

Row (ULONG) - input
Starting row location.

Starting row of the field to read, where 0 is the top row.

Column (ULONG) - input
Starting column location.

Starting column of the field to read, where 0 is the leftmost column.

VioHandle (HVIO) - input
Vio presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioReadCharStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW

359
421
436

ERROR_VIO_COL
ERROR_VIO_INVALID_PARMS
ERROR_VIO_INVALID_HANDLE

VioReadCharStr - Remarks

If a string read comes to the end of the line and is not complete, it continues at the beginning of the next line. If the read comes to the end of the screen and is not complete, it terminates, and the length is set to the number of characters read.

VioReadCharStr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioReadCellStr

VioReadCellStr - Syntax

VioReadCellStr reads a string of character-attribute pairs (cells) from the screen, starting at the specified location.

```
#define INCL_VIO
#include <os2.h>

PCH      CellStr; /* Cell string buffer. */
PULONG   Length; /* Length of cell string buffer. */
ULONG    Row;     /* Starting row location. */
ULONG    Column;  /* Starting column location. */
HVIO     VioHandle; /* Vio presentation-space handle. */
APIRET   rc;      /* Return code. */

rc = VioReadCellStr(CellStr, Length, Row,
                    Column, VioHandle);
```

VioReadCellStr Parameter - CellStr

CellStr (PCH) - output
Cell string buffer.

Address of the buffer where the cell string is returned.

VioReadCellStr Parameter - Length

Length (PULONG) - in/out
Length of cell string buffer.

Address of the buffer length in bytes. Length must take into account that each character-attribute entry in the buffer is 2 or 4 bytes. If the length of the buffer is not sufficient, the last entry is not complete.

VioReadCellStr Parameter - Row

Row (ULONG) - input
Starting row location.

Starting row of the field to read, where 0 is the top row.

VioReadCellStr Parameter - Column

Column (ULONG) - input
Starting column location.

Starting column of the field to read, where 0 is the leftmost column.

VioReadCellStr Parameter - VioHandle

VioHandle (HVIO) - input
Vio presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case it must be the value returned by [VioCreatePS](#).

VioReadCellStr Return Value - rc

rc (APIRET) - returns
Return code.

VioReadCellStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE

358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioReadCellStr - Parameters

CellStr (PCH) - output
Cell string buffer.

Address of the buffer where the cell string is returned.

Length (PULONG) - in/out
Length of cell string buffer.

Address of the buffer length in bytes. Length must take into account that each character-attribute entry in the buffer is 2 or 4 bytes. If the length of the buffer is not sufficient, the last entry is not complete.

Row (ULONG) - input
Starting row location.

Starting row of the field to read, where 0 is the top row.

Column (ULONG) - input
Starting column location.

Starting column of the field to read, where 0 is the leftmost column.

VioHandle (HVIO) - input
Vio presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioReadCellStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioReadCellStr - Remarks

If a string read comes to the end of the line and is not complete, it continues at the beginning of the next line. If the read comes to the end of the screen and is not complete, it terminates and the length is set to the length of the buffer that was filled.

VioReadCellStr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

VioSavRedrawWait

VioSavRedrawWait - Syntax

VioSavRedrawWait notifies a graphics mode application when it must save or redraw its screen image.

```
#define INCL_VIO
#include <os2.h>

ULONG      SaveRedrawIndic; /* Save/redraw indicator. */
PULONG     NotifyType;     /* Notify type (returned). */
HVIO       VioHandle;      /* Reserved. Must be 0. */
APIRET     rc;             /* Return code. */

rc = VioSavRedrawWait(SaveRedrawIndic, NotifyType,
                     VioHandle);
```

VioSavRedrawWait Parameter - SaveRedrawIndic

SaveRedrawIndic (ULONG) - input
Save/redraw indicator.

Indicates which events the application is waiting for:

Value	Definition
0	The session manager notifies the application for both save and redraw operations.
1	The session manager notifies the application for redraw operations only.

VioSavRedrawWait Parameter - NotifyType

NotifyType (PULONG) - output
Notify type (returned).

Address that specifies the operation to be performed by the application upon return from VioSavRedrawWait:

Value	Definition
0	Save screen image.
1	Restore screen image.

VioSavRedrawWait Parameter - VioHandle

VioHandle (HVIO) - input
Reserved. Must be 0.

VioSavRedrawWait Return Value - rc

rc (APIRET) - returns
Return code.

VioSavRedrawWait returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
423	ERROR_VIO_RETURN
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioSavRedrawWait - Parameters

SaveRedrawIndic (ULONG) - input
Save/redraw indicator.

Indicates which events the application is waiting for:

Value	Definition
0	The session manager notifies the application for both save and redraw operations.
1	The session manager notifies the application for redraw operations only.

NotifyType (PULONG) - output
Notify type (returned).

Address that specifies the operation to be performed by the application upon return from VioSavRedrawWait:

Value	Definition
0	Save screen image.
1	Restore screen image.

VioHandle (HVIO) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioSavRedrawWait returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
423	ERROR_VIO_RETURN
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioSavRedrawWait - Remarks

The OS/2 operating system uses `VioSavRedrawWait` to notify a graphics-mode application to save or restore its screen image at screen switch time. The application in the outgoing foreground session is notified to perform a save. The application in the incoming foreground session is notified to perform a restore. The application must perform the action requested and immediately reissue `VioSavRedrawWait`. When an application performs a save, it saves its physical display buffer, video mode, and any other information the application needs to completely redraw its screen at restore time.

Only one process per session can issue `VioSavRedrawWait`. The process that first issues it becomes the owner of the function.

A text-mode application must issue `VioSavRedrawWait` only if the application writes directly to the registers on the display adapter. Assuming `VioSavRedrawWait` is not issued by a text-mode application, the OS/2 operating system performs the required saves and restores.

An application that issues `VioSavRedrawWait` might also need to issue `VioModeWait`. This would allow the application to be notified when it must restore its mode at the completion of an application or hard-error pop-up. See `VioModeWait` for more information. Two application threads would be required to perform these operations, in this case.

At the time a `VioSavRedrawWait` thread is notified, the session is in transition to or from the background. Although the session's official status is background, any selector to the physical display buffer previously obtained by the `VioSavRedrawWait` process (through `VioGetPhysBuf`) is valid, at this time. The physical display buffer must be accessed without issuing `VioScrLock`. Because the session's official status is background, any thread waits if it issues `VioScrLock` with the "wait if unsuccessful" option.

An application containing a `VioSavRedrawWait` thread should be designed so that the process does not cause any hard errors while the `VioSavRedrawWait` thread is running; otherwise, a system lockout might occur.

An application's `VioSavRedrawWait` thread might be notified to perform a restore before it is notified to perform a save. This happens if the application was running in the background the first time it issued `VioSavRedrawWait`. The return from this function call provides the notification. The thread that issues the call performs the save or redraw and then reissues `VioSavRedrawWait` to wait until its screen image must be saved or redrawn again.

VioSavRedrawWait - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioSavRedrawUndo

VioSavRedrawUndo - Syntax

`VioSavRedrawUndo` allows one thread within a process to cancel a `VioSavRedrawWait` issued by another thread within the same process.

```
#define INCL_VIO
#include <os2.h>

ULONG      OwnerIndic; /* Ownership indicator */
ULONG      KillIndic;  /* Terminate indicator */
HVIO       VioHandle;  /* Reserved. Must be 0. */
APIRET     rc;          /* Return code. */

rc = VioSavRedrawUndo(OwnerIndic, KillIndic,
                     VioHandle);
```

VioSavRedrawUndo Parameter - OwnerIndic

OwnerIndic (ULONG) - input
Ownership indicator

Indicates whether the thread issuing VioSavRedrawUndo wants ownership of VioSavRedrawUndo to be reserved for its process.

	Value	Definition
0		Reserve ownership.
1		Give up ownership.

VioSavRedrawUndo Parameter - KillIndic

KillIndic (ULONG) - input
Terminate indicator

Indicates whether the thread (with the outstanding VioSavRedrawUndo) should return an error code or be terminated.

	Value	Definition
0		Return error code.
1		Terminate thread.

VioSavRedrawUndo Parameter - VioHandle

VioHandle (HVIO) - input
Reserved. Must be 0.

VioSavRedrawUndo Return Value - rc

rc (APIRET) - returns
Return code.

VioSavRedrawUndo returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
428	ERROR_VIO_NO_SAVE_RESTORE_THD
430	ERROR_VIO_ILLEGAL_DURING_POPUP

VioSavRedrawUndo - Parameters

OwnerIndic (ULONG) - input
Ownership indicator

Indicates whether the thread issuing VioSavRedrawUndo wants ownership of VioSavRedrawUndo to be reserved for its process.

Value	Definition
0	Reserve ownership.
1	Give up ownership.

KillIndic (ULONG) - input
Terminate indicator

Indicates whether the thread (with the outstanding VioSavRedrawUndo) should return an error code or be terminated.

Value	Definition
0	Return error code.
1	Terminate thread.

VioHandle (HVIO) - input
Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioSavRedrawUndo returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
428	ERROR_VIO_NO_SAVE_RESTORE_THD
430	ERROR_VIO_ILLEGAL_DURING_POPUP

VioSavRedrawUndo - Remarks

The issuing thread can reserve ownership of [VioSavRedrawWait](#) for its process or give it up. The thread whose [VioSavRedrawWait](#) was canceled is optionally terminated. VioSavRedrawUndo can be issued only by a thread within the same process that owns [VioSavRedrawWait](#).

VioSavRedrawUndo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioScrLock

VioScrLock - Syntax

VioScrLock requests ownership of (locks) the physical display buffer.

```

#define INCL_VIO
#include <os2.h>

ULONG      WaitFlag;    /* Wait until screen I/O can take place. */
PUCHAR     Status;      /* Address of the lock status. */
HVIO       VioHandle;    /* VIO presentation-space handle. */
APIRET     rc;           /* Return code. */

rc = VioScrLock(WaitFlag, Status, VioHandle);

```

VioScrLock Parameter - WaitFlag

WaitFlag (ULONG) - input

Wait until screen I/O can take place.

Indicates whether the process should block until the screen I/O can take place.

Value	Definition
0	Return if screen I/O not available.
1	Wait until screen I/O is available.

VioScrLock Parameter - Status

Status (PUCHAR) - output

Address of the lock status.

The lock status may be one of the following values:

Value	Definition
0	Lock successful.
1	Lock unsuccessful (in the case of no wait). Status is returned only when AX = 0. Status = 1 can be returned only when WaitFlag = 0.

VioScrLock Parameter - VioHandle

VioHandle (HVIO) - input

VIO presentation-space handle.

Reserved. Must be 0.

VioScrLock Return Value - rc

rc (APIRET) - returns

Return code.

VioScrLock returns one of the following values:

0	NO_ERROR
366	ERROR_VIO_WAIT_FLAG
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
434	ERROR_VIO_LOCK
436	ERROR_VIO_INVALID_HANDLE

VioScrLock - Parameters

WaitFlag (ULONG) - input

Wait until screen I/O can take place.

Indicates whether the process should block until the screen I/O can take place.

Value	Definition
0	Return if screen I/O not available.
1	Wait until screen I/O is available.

Status (PUCHAR) - output

Address of the lock status.

The lock status may be one of the following values:

Value	Definition
0	Lock successful.
1	Lock unsuccessful (in the case of no wait). Status is returned only when AX = 0. Status = 1 can be returned only when WaitFlag = 0.

VioHandle (HVIO) - input

VIO presentation-space handle.

Reserved. Must be 0.

rc (APIRET) - returns

Return code.

VioScrLock returns one of the following values:

0	NO_ERROR
366	ERROR_VIO_WAIT_FLAG
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
434	ERROR_VIO_LOCK
436	ERROR_VIO_INVALID_HANDLE

VioScrLock - Remarks

VioScrLock permits a process to determine if I/O to the physical screen buffer can take place. This prevents the process from writing to the physical buffer when the process is in the background. Processes must cooperate with the system in coordinating screen accesses.

Screen switching is disabled while the screen lock is in place. If a screen switch is suspended by a screen lock, and if the application holding the lock does not issue [VioScrUnLock](#) within a system-defined time limit, the screen switch occurs, and the process holding the lock is frozen in the background. A process should yield the screen lock as soon as possible, to avoid being frozen when running in the background. The timeout on the lock does not begin until a screen switch is requested.

When the screen lock is in effect and another thread in the same or different process (in the same session) issues VioScrLock, the second thread receives an error code. VioScrUnLock must be issued by the same thread that issued VioScrLock.

VioScrLock - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioScrollDown

VioScrollDown - Syntax

VioScrollDown scrolls the entire display buffer (or area specified within the display buffer) down.

```
#define INCL_VIO
#include <os2.h>

ULONG      TopRow;      /* Top row to be scrolled. */
ULONG      LeftCol;     /* Left column to be scrolled. */
ULONG      BotRow;      /* Bottom row to be scrolled. */
ULONG      RightCol;    /* Right column to be scrolled. */
ULONG      Lines;       /* Number of lines. */
PBYTE      Cell;        /* Cell to be written. */
HVIO       VioHandle;   /* Vio presentation-space handle. */
APIRET     rc;          /* Return code. */

rc = VioScrollDown(TopRow, LeftCol, BotRow,
                   RightCol, Lines, Cell, VioHandle);
```

VioScrollDown Parameter - TopRow

TopRow (ULONG) - input
Top row to be scrolled.

VioScrollDown Parameter - LeftCol

LeftCol (ULONG) - input
Left column to be scrolled.

VioScrollDown Parameter - BotRow

BotRow (ULONG) - input
Bottom row to be scrolled.

VioScrollDown Parameter - RightCol

RightCol (ULONG) - input
Right column to be scrolled.

VioScrollDown Parameter - Lines

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the top of the screen area being scrolled. If 0 is specified, no lines are scrolled.

VioScrollDown Parameter - Cell

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioScrollDown Parameter - VioHandle

VioHandle (HVIO) - input
Vio presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioScrollDown Return Value - rc

rc (APIRET) - returns
Return code.

VioScrollDown returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioScrollDown - Parameters

TopRow (ULONG) - input
Top row to be scrolled.

LeftCol (ULONG) - input
Left column to be scrolled.

BotRow (ULONG) - input
Bottom row to be scrolled.

RightCol (ULONG) - input
Right column to be scrolled.

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the top of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioHandle (HVIO) - input
Vio presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioScrollDown returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioScrollDown - Remarks

TopRow = 0 and *LeftCol* = 0 identifies the top-left corner of the screen.

If a value greater than the maximum value is specified for *TopRow* , *LeftCol* , *BotRow* , *RightCol* , or *Lines* , the maximum value for that parameter is used.

If *TopRow* and *LeftCol* = 0, and if *BotRow* , *RightCol* , and *Lines* are greater than the screen lines, the entire screen is filled with the character-attribute pair defined by *Cell* .

VioScrollDown - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioScrollLeft

VioScrollLeft - Syntax

VioScrollLeft scrolls the entire display buffer (or area specified within the display buffer) to the left.

```
#define INCL_VIO
#include <os2.h>

ULONG      TopRow;      /* Top row to be scrolled. */
ULONG      LeftCol;     /* Left column to be scrolled. */
ULONG      BotRow;      /* Bottom row to be scrolled. */
ULONG      RightCol;    /* Right column to be scrolled. */
ULONG      Lines;       /* Number of lines. */
PBYTE      Cell;        /* Cell to be written. */
HVIO       VioHandle;   /* VIO presentation-space handle. */
APIRET     rc;          /* Return code. */

rc = VioScrollLeft(TopRow, LeftCol, BotRow,
                   RightCol, Lines, Cell, VioHandle);
```

VioScrollLeft Parameter - TopRow

TopRow (ULONG) - input
Top row to be scrolled.

VioScrollLeft Parameter - LeftCol

LeftCol (ULONG) - input
Left column to be scrolled.

VioScrollLeft Parameter - BotRow

BotRow (ULONG) - input
Bottom row to be scrolled.

VioScrollLeft Parameter - RightCol

RightCol (ULONG) - input
Right column to be scrolled.

VioScrollLeft Parameter - Lines

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the right of the screen area being scrolled. If 0 is specified, no lines are scrolled.

VioScrollLeft Parameter - Cell

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioScrollLeft Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioScrollLeft Return Value - rc

rc (APIRET) - returns
Return code.

VioScrollLeft returns one of the following values:

0 NO_ERROR

355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioScrollLeft - Parameters

TopRow (ULONG) - input
Top row to be scrolled.

LeftCol (ULONG) - input
Left column to be scrolled.

BotRow (ULONG) - input
Bottom row to be scrolled.

RightCol (ULONG) - input
Right column to be scrolled.

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the right of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioScrollLeft returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioScrollLeft - Remarks

TopRow = 0 and *LeftCol* = 0 identify the top-left corner of the screen.

If a value greater than the maximum value is specified for *TopRow* , *LeftCol* , *BotRow* , *RightCol* , or *Lines* , the maximum value for that parameter is used.

If *TopRow* and *LeftCol* = 0, and if *BotRow* , *RightCol* , and *Lines* are greater than the screen lines, the entire screen is filled with the character-attribute pair defined by *Cell* .

VioScrollLeft - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioScrollRight

VioScrollRight - Syntax

VioScrollRight scrolls the entire display buffer (or area specified within the display buffer) to the right.

```
#define INCL_VIO
#include <os2.h>

ULONG      TopRow;      /* Top row to be scrolled. */
ULONG      LeftCol;     /* Left column to be scrolled. */
ULONG      BotRow;      /* Bottom row to be scrolled. */
ULONG      RightCol;    /* Right column to be scrolled. */
ULONG      Lines;       /* Number of lines. */
PBYTE      Cell;        /* Cell to be written. */
HVIO       VioHandle;   /* VIO presentation-space handle. */
APIRET     rc;          /* Return code. */

rc = VioScrollRight(TopRow, LeftCol, BotRow,
                    RightCol, Lines, Cell, VioHandle);
```

VioScrollRight Parameter - TopRow

TopRow (ULONG) - input
Top row to be scrolled.

VioScrollRight Parameter - LeftCol

LeftCol (ULONG) - input
Left column to be scrolled.

VioScrollRight Parameter - BotRow

BotRow (ULONG) - input
Bottom row to be scrolled.

VioScrollRight Parameter - RightCol

RightCol (ULONG) - input
Right column to be scrolled.

VioScrollRight Parameter - Lines

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the left of the screen area being scrolled. If 0 is specified, no lines are scrolled.

VioScrollRight Parameter - Cell

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioScrollRight Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioScrollRight Return Value - rc

rc (APIRET) - returns
Return code.

VioScrollRight returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE

358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioScrollRight - Parameters

TopRow (ULONG) - input
Top row to be scrolled.

LeftCol (ULONG) - input
Left column to be scrolled.

BotRow (ULONG) - input
Bottom row to be scrolled.

RightCol (ULONG) - input
Right column to be scrolled.

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the left of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioScrollRight returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioScrollRight - Remarks

TopRow = 0 and *LeftCol* = 0 identify the top-left corner of the screen.

If a value greater than the maximum value is specified for *TopRow* , *LeftCol* , *BotRow* , *RightCol* , or *Lines* , the maximum value for that parameter is used.

If *TopRow* and *LeftCol* = 0, and if *BotRow* , *RightCol* , and *Lines* are greater than the screen lines, the entire screen is filled with the character-attribute pair defined by *Cell* .

VioScrollRight - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioScrollUp

VioScrollUp - Syntax

VioScrollUp scrolls up the entire display buffer (or area specified within the display buffer).

```
#define INCL_VIO
#include <os2.h>

ULONG      TopRow;      /* Top row to be scrolled. */
ULONG      LeftCol;     /* Left column to be scrolled. */
ULONG      BotRow;      /* Bottom row to be scrolled. */
ULONG      RightCol;    /* Right column to be scrolled. */
ULONG      Lines;       /* Number of lines. */
PBYTE      Cell;        /* Cell to be written. */
HVIO       VioHandle;   /* VIO presentation-space handle. */
APIRET     rc;          /* Return code. */

rc = VioScrollUp(TopRow, LeftCol, BotRow,
                 RightCol, Lines, Cell, VioHandle);
```

VioScrollUp Parameter - TopRow

TopRow (ULONG) - input
Top row to be scrolled.

VioScrollUp Parameter - LeftCol

LeftCol (ULONG) - input
Left column to be scrolled.

VioScrollUp Parameter - BotRow

BotRow (ULONG) - input
Bottom row to be scrolled.

VioScrollUp Parameter - RightCol

RightCol (ULONG) - input
Right column to be scrolled.

VioScrollUp Parameter - Lines

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the bottom of the screen area being scrolled. If 0 is specified, no lines are scrolled.

VioScrollUp Parameter - Cell

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioScrollUp Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioScrollUp Return Value - rc

rc (APIRET) - returns
Return code.

VioScrollUp returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL

421
436

ERROR_VIO_INVALID_PARMS
ERROR_VIO_INVALID_HANDLE

VioScrollUp - Parameters

TopRow (ULONG) - input
Top row to be scrolled.

LeftCol (ULONG) - input
Left column to be scrolled.

BotRow (ULONG) - input
Bottom row to be scrolled.

RightCol (ULONG) - input
Right column to be scrolled.

Lines (ULONG) - input
Number of lines.

Number of lines to be inserted at the bottom of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell (PBYTE) - input
Cell to be written.

Address of the character-attribute pair (2 or 4 bytes) used as a fill character on inserted lines.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioScrollUp returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioScrollUp - Remarks

TopRow = 0 and *LeftCol* = 0 identify the top-left corner of the screen.

If a value greater than the maximum value is specified for *TopRow* , *LeftCol* , *BotRow* , *RightCol* , or *Lines* , the maximum value for that parameter is used.

If *TopRow* and *LeftCol* = 0, and if *BotRow* , *RightCol* , and *Lines* are greater than the screen lines, the entire screen is filled with the character-attribute pair defined by *Cell* .

VioScrollUp - Topics

Select an item:

VioScrUnlock

VioScrUnlock - Syntax

VioScrUnlock releases ownership of (unlocks) the physical display buffer.

```
#define INCL_VIO
#include <os2.h>

HVIO      VioHandle; /* VIO presentation-space handle. */
APIRET    rc;        /* Return code. */

rc = VioScrUnlock(VioHandle);
```

VioScrUnlock Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

Reserved. Must be 0.

VioScrUnlock Return Value - rc

rc (APIRET) - returns
Return code.

VioScrUnlock returns one of the following values:

0	NO_ERROR
367	ERROR_VIO_UNLOCK
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioScrUnlock - Parameters

VioHandle (HVIO) - input
VIO presentation-space handle.

Reserved. Must be 0.

rc (APIRET) - returns
Return code.

VioScrUnlock returns one of the following values:

0	NO_ERROR
367	ERROR_VIO_UNLOCK
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioScrUnlock - Remarks

This call releases the screen lock that is set by [VioScrLock](#). The VioScrUnlock call must be issued by the same thread that issued VioScrLock.

VioScrUnlock - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioSetAnsi

VioSetAnsi - Syntax

VioSetAnsi activates or deactivates ANSI support.

```
#define INCL_VIO
#include <os2.h>

ULONG      Indicator; /* On/Off indicator. */
HVIO      VioHandle; /* VIO presentation-space handle. */
APIRET     rc;        /* Return code. */

rc = VioSetAnsi(Indicator, VioHandle);
```

VioSetAnsi Parameter - Indicator

Indicator (ULONG) - input
On/Off indicator.

Equals 1 to activate ANSI support, or 0 to deactivate ANSI.

VioSetAnsi Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioSetAnsi Return Value - rc

rc (APIRET) - returns
Return code.

VioSetAnsi returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioSetAnsi - Parameters

Indicator (ULONG) - input
On/Off indicator.

Equals 1 to activate ANSI support, or 0 to deactivate ANSI.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioSetAnsi returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioSetAnsi - Remarks

For ANSI support, "ON" is the default.

VioSetAnsi - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioSetCp

VioSetCp - Syntax

VioSetCp sets the code page used to display text data on the screen for the specified handle.

```
#define INCL_VIO
#include <os2.h>

ULONG      Reserved;    /* Reserved. Must be 0. */
USHORT     CodePageID;  /* Code-page ID. */
HVIO       VioHandle;   /* VIO presentation-space handle. */
APIRET     rc;          /* Return code. */

rc = VioSetCp(Reserved, CodePageID, VioHandle);
```

VioSetCp Parameter - Reserved

Reserved (ULONG) - input
Reserved. Must be 0.

VioSetCp Parameter - CodePageID

CodePageID (USHORT) - input
Code-page ID.

The *CodePageID* must be a known code page. Note that the values 0, -1, and -2, supported by OS/2 2.x, will cause an error in OS/2 Warp (PowerPC Edition)

VioSetCp Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioSetCp Return Value - rc

rc (APIRET) - returns
Return code.

VioSetCp returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
469	ERROR_VIO_BAD_CP

VioSetCp - Parameters

Reserved (ULONG) - input
Reserved. Must be 0.

CodePageID (USHORT) - input
Code-page ID.

The *CodePageID* must be a known code page. Note that the values 0, -1, and -2, supported by OS/2 2.x, will cause an error in OS/2 Warp (PowerPC Edition)

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioSetCp returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
469	ERROR_VIO_BAD_CP

VioSetCp - Remarks

The specified code page applies to all new characters. How VioSetCp acts on characters already in the video buffer is undefined.

VioSetCp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioSetCurPos

VioSetCurPos - Syntax

VioSetCurPos sets the cursor's coordinates on the screen.

```
#define INCL_VIO
#include <os2.h>

ULONG      Row;
USHORT     Column;
HVIO       VioHandle; /* VIO presentation-space handle. */
APIRET     rc;        /* Return code. */

rc = VioSetCurPos(Row, Column, VioHandle);
```

VioSetCurPos Parameter - Row

Row (ULONG) - input
New cursor row position, where 0 is the top row.

VioSetCurPos Parameter - Column

Column (USHORT) - input

New cursor column position, where 0 is the leftmost column.

VioSetCurPos Parameter - VioHandle

VioHandle (HVIO) - input

VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioSetCurPos Return Value - rc

rc (APIRET) - returns

Return code.

VioSetCurPos returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioSetCurPos - Parameters

Row (ULONG) - input

New cursor row position, where 0 is the top row.

Column (USHORT) - input

New cursor column position, where 0 is the leftmost column.

VioHandle (HVIO) - input

VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns

Return code.

VioSetCurPos returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioSetCurPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

VioSetCurType

VioSetCurType - Syntax

VioSetCurType sets the cursor type.

```
#define INCL_VIO
#include <os2.h>

PVIOCURSORINFO    CursorData; /* Cursor characteristics. */
HVIO               VioHandle; /* VIO presentation-space handle. */
APIRET            rc;         /* Return code. */

rc = VioSetCurType(CursorData, VioHandle);
```

VioSetCurType Parameter - CursorData

CursorData ([PVIOCURSORINFO](#)) - input
Cursor characteristics.

Address of the cursor-characteristics structure.

VioSetCurType Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioSetCurType Return Value - rc

rc (APIRET) - returns
Return code.

VioSetCurType returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
356	ERROR_VIO_WIDTH
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

VioSetCurType - Parameters

CursorData ([PVIOCURSORINFO](#)) - input
Cursor characteristics.

Address of the cursor-characteristics structure.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioSetCurType returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
356	ERROR_VIO_WIDTH
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

VioSetCurType - Remarks

To set the cursor start line (*yStart*) and the cursor end line (*cEnd*) independent of the number of scan lines for each character cell, you can specify these parameters as percentages. The OS/2 operating system then calculates the physical start and end scan lines, respectively, by multiplying the percentage specified for the parameter by the total number of scan lines in the character cell and rounding to the nearest scan line. Percentages are specified as negative values (or 0) in the range 0 through -100. Specifying *yStart* = -90 and *cEnd* = -100 requests a cursor that occupies the bottom 10% of the character cell.

The actual appearance of the cursor is hardware dependent. The video hardware might not support the specified parameters.

VioSetCurType - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

VioSetDeviceCellSize

VioSetDeviceCellSize - Syntax

VioSetDeviceCellSize sets the size of the current character cell in pels.

```
#define INCL_VIO
#include <os2.h>

ULONG      Height; /* The height of the character cell in pels. */
ULONG      Width;  /* The width of the character cell in pels. */
HVIO       hvps;   /* VIO presentation-space handle. */
APIRET     rc;     /* Return code. */

rc = VioSetDeviceCellSize(Height, Width, hvps);
```

VioSetDeviceCellSize Parameter - Height

Height (ULONG) - input
The height of the character cell in pels.

VioSetDeviceCellSize Parameter - Width

Width (ULONG) - input
The width of the character cell in pels.

VioSetDeviceCellSize Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session or a value returned by apiref refid='viocrps' form='textonly'..

VioSetDeviceCellSize Return Value - rc

rc (APIRET) - returns
Return code.

VioSetDeviceCellSize returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioSetDeviceCellSize - Parameters

Height (ULONG) - input
The height of the character cell in pels.

Width (ULONG) - input
The width of the character cell in pels.

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session or a value returned by apiref refid='viocrps' form='textonly'..

rc (APIRET) - returns
Return code.

VioSetDeviceCellSize returns one of the following values:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioSetDeviceCellSize - Remarks

If the device does not support the specified cell size, the cell size closest to the specified size is used. VioGetDeviceCellSize can be used to find the actual size selected. In OS/2 2.x, *hvps* cannot be 0.

VioSetDeviceCellSize - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioSetMode

VioSetMode - Syntax

VioSetMode sets the mode of the display.

```
#define INCL_VIO
#include <os2.h>

PVIOMODEINFO    ModeData;    /* Mode characteristics. */
HVIO            VioHandle;    /* VIO presentation-space handle. */
APIRET          rc;          /* Return code. */

rc = VioSetMode(ModeData, VioHandle);
```

VioSetMode Parameter - ModeData

ModeData (PVIOMODEINFO) - input
Mode characteristics.

Address of the mode characteristics structure.

VioSetMode Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioSetMode Return Value - rc

rc (APIRET) - returns
Return code.

VioSetMode returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH
467	ERROR_VIO_FONT

VioSetMode - Parameters

ModeData ([PVIOMODEINFO](#)) - input
Mode characteristics.

Address of the mode characteristics structure.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioSetMode returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH
467	ERROR_VIO_FONT

VioSetMode - Remarks

VioSetMode initializes the cursor position and type.

VioSetMode does not clear the screen if the new and old modes are compatible. To clear the screen, use one of the VioScrollxx calls.

Assuming that no target display configuration for VioSetMode is selected, the mode is set on the primary configuration. If the primary configuration does not support the specified mode, the mode is set on the secondary configuration.

VioSetMode - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioSetOrigin

VioSetOrigin - Syntax

VioSetOrigin sets the position at which the presentation space maps to the window.

```
#define INCL_VIO
#include <os2.h>

ULONG      Row;      /* The top-most row shown in the window */
ULONG      Column;    /* The left-most column shown in the window. */
HVIO       hvps;      /* VIO presentation-space handle. */
APIRET     rc;        /* Return code. */

rc = VioSetOrigin(Row, Column, hvps);
```

VioSetOrigin Parameter - Row

Row (ULONG) - input
The top-most row shown in the window

VioSetOrigin Parameter - Column

Column (ULONG) - input
The left-most column shown in the window.

VioSetOrigin Parameter - hvps

hvps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session or a value returned by [VioCreatePS](#).

VioSetOrigin Return Value - rc

rc (APIRET) - returns
Return code.

VioSetOrigin returns one of the following values:

0	NO_ERROR
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioSetOrigin - Parameters

Row (ULONG) - input
The top-most row shown in the window

Column (ULONG) - input
The left-most column shown in the window.

hyps (HVIO) - input
VIO presentation-space handle.

This is either 0 to indicate the default VIO session or a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioSetOrigin returns one of the following values:

0	NO_ERROR
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioSetOrigin - Remarks

VioSetOrigin is used when the presentation space is larger than the window size to control which part of the presentation space is displayed. It does not, itself, cause any output to be displayed.

In OS/2 2.x, *hyps* cannot be 0.

VioSetOrigin - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioSetState

VioSetState - Syntax

VioSetState performs one of the following functions: set palette registers, set the overscan (border) color, set the blink/background intensity switch, set color registers, set the underline location, or set the target VioSetMode display configuration.

```
#define INCL_VIO
#include <os2.h>
```

```

PVIOD      RequestBlock;
HVIO       VioHandle;      /* VIO presentation-space handle */
APIRET     rc;             /* Return code. */

rc = VioSetState(RequestBlock, VioHandle);

```

VioSetState Parameter - RequestBlock

RequestBlock (PVIOD) - input

Address of the video state structures (consisting of six different structures, depending on the request type):

	Type	Definition
0		Set palette registers
1		Set overscan (border) color
2		Set blink/background intensity switch
3		Set color registers
4		Reserved
5		Set underline location
6		Set target VioSetMode display configuration

The six structures, depending on request type, are:

- [VIOPALSTATE](#)
- [VIOOVERSCAN](#)
- [VIOINTENSITY](#)
- [VIOCOLORREG](#)
- [VIOSETULINELOC](#)
- [VIOSETTARGET](#)

VioSetState Parameter - VioHandle

VioHandle (HVIO) - input

VIO presentation-space handle

Reserved. Must be 0.

VioSetState Return Value - rc

rc (APIRET) - returns

Return code.

VioSetState returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioSetState - Parameters

RequestBlock (PVIOD) - input

Address of the video state structures (consisting of six different structures, depending on the request type):

Type	Definition
0	Set palette registers
1	Set overscan (border) color
2	Set blink/background intensity switch
3	Set color registers
4	Reserved
5	Set underline location
6	Set target VioSetMode display configuration

The six structures, depending on request type, are:

- [VIOPALSTATE](#)
- [VIOOVERSCAN](#)
- [VIOINTENSITY](#)
- [VIOCOLORREG](#)
- [VIOSETULINELOC](#)
- [VIOSETTARGET](#)

VioHandle (HVIO) - input

VIO presentation-space handle

Reserved. Must be 0.

rc (APIRET) - returns

Return code.

VioSetState returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH

VioSetState - Remarks

VioSetState allows setting of hardware-dependent features. Not all video hardware will honor these settings.

VioSetState - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

VioShowBuf

VioShowBuf - Syntax

VioShowBuf updates the physical display buffer with the logical video buffer (LVB).

```
#define INCL_VIO
#include <os2.h>

ULONG      OffSet;      /* Offset into the LVB. */
ULONG      Length;
HVIO       VioHandle;   /* VIO presentation-space handle. */
APIRET     rc;          /* Return code */

rc = VioShowBuf(Offset, Length, VioHandle);
```

VioShowBuf Parameter - OffSet

OffSet (ULONG) - input
Offset into the LVB.

Starting offset, within the LVB, where the update to the screen is to start.

VioShowBuf Parameter - Length

Length (ULONG) - input
Length of the area to be updated to the screen.

VioShowBuf Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioShowBuf Return Value - rc

rc (APIRET) - returns
Return code

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE

421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioShowBuf - Parameters

OffSet (ULONG) - input
Offset into the LVB.

Starting offset, within the LVB, where the update to the screen is to start.

Length (ULONG) - input
Length of the area to be updated to the screen.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE

VioShowBuf - Remarks

VioShowBuf is ignored unless it is issued by a process that is currently executing in the foreground or in a window.

VioShowBuf - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioShowPS

VioShowPS - Syntax

Update the display of the VIO presentation space.

```
#define INCL_VIO
#include <os2.h>

ULONG      Depth; /* Depth. */
ULONG      Width; /* Width. */
ULONG      Cell;  /* Offset to first updated cell. */
HVIO       hvps;  /* VIO presentation handle. */
APIRET     rc;    /* Return code */

rc = VioShowPS(Depth, Width, Cell, hvps);
```

VioShowPS Parameter - Depth

Depth (ULONG) - input
Depth.

The depth of the updated rectangle.

VioShowPS Parameter - Width

Width (ULONG) - input
Width.

The width of the updated rectangle.

VioShowPS Parameter - Cell

Cell (ULONG) - input
Offset to first updated cell.

The offset to the first updated cell. The offset of the top-left corner is zero.

VioShowPS Parameter - hvps

hvps (HVIO) - input
VIO presentation handle.

VIO presentation space handle. This is either zero to indicate the default VIO session or a value returned by [VioCreatePS](#).

VioShowPS Return Value - rc

rc (APIRET) - returns
Return code

Return code descriptions are:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioShowPS - Parameters

Depth (ULONG) - input
Depth.

The depth of the updated rectangle.

Width (ULONG) - input
Width.

The width of the updated rectangle.

Cell (ULONG) - input
Offset to first updated cell.

The offset to the first updated cell. The offset of the top-left corner is zero.

hvps (HVIO) - input
VIO presentation handle.

VIO presentation space handle. This is either zero to indicate the default VIO session or a value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code

Return code descriptions are:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioShowPS - Remarks

This call is used to specify that part or all of the presentation space the logical buffer needs to be redrawn.

This call has the same function as [VioShowBuf](#), but the area to update is specified differently.

VioShowPS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioWrtCellStr

VioWrtCellStr - Syntax

VioWrtCellStr writes a string of character-attribute pairs (cells) to the display.

```
#define INCL_VIO
#include <os2.h>

ULONG      CellStr;    /* String to be written. */
ULONG      Length;    /* Length of string. */
ULONG      Row;       /* Starting row position for output. */
ULONG      Column;    /* Starting column position for output. */
HVIO       VioHandle; /* VIO presentation-space handle. */
APIRET     rc;        /* Return code. */

rc = VioWrtCellStr(CellStr, Length, Row, Column,
                  VioHandle);
```

VioWrtCellStr Parameter - CellStr

CellStr (ULONG) - input
String to be written.

Address of the string of character-attribute cells to be written.

VioWrtCellStr Parameter - Length

Length (ULONG) - input
Length of string.

Length, in bytes, of the string to be written. Each character-attribute cell is 2 or 4 bytes.

VioWrtCellStr Parameter - Row

Row (ULONG) - input
Starting row position for output.

VioWrtCellStr Parameter - Column

Column (ULONG) - input
Starting column position for output.

VioWrtCellStr Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioWrtCellStr Return Value - rc

rc (APIRET) - returns
Return code.

VioWrtCellStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtCellStr - Parameters

CellStr (ULONG) - input
String to be written.

Address of the string of character-attribute cells to be written.

Length (ULONG) - input
Length of string.

Length, in bytes, of the string to be written. Each character-attribute cell is 2 or 4 bytes.

Row (ULONG) - input
Starting row position for output.

Column (ULONG) - input
Starting column position for output.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioWrtCellStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtCellStr - Remarks

If a string write gets to the end of the line and is not complete, it continues at the beginning of the next line. If the write gets to the end of the screen, it terminates.

VioWrtCellStr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioWrtCharStr

VioWrtCharStr - Syntax

VioWrtCharStr writes a character string to the display.

```
#define INCL_VIO
#include <os2.h>

PCH      CharStr;    /* String to be written. */
ULONG    Length;     /* Length of character string. */
```

```
ULONG      Row;          /* Starting row position. */
ULONG      Column;       /* Starting column position. */
HVIO       VioHandle;    /* VIO presentation-space handle. */
APIRET     rc;           /* Return code. */
```

```
rc = VioWrtCharStr(CharStr, Length, Row, Column,
                  VioHandle);
```

VioWrtCharStr Parameter - CharStr

CharStr (PCH) - input
String to be written.

Address of the character string to be written.

VioWrtCharStr Parameter - Length

Length (ULONG) - input
Length of character string.

Length, in bytes, of the character string.

VioWrtCharStr Parameter - Row

Row (ULONG) - input
Starting row position.

VioWrtCharStr Parameter - Column

Column (ULONG) - input
Starting column position.

VioWrtCharStr Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioWrtCharStr Return Value - rc

rc (APIRET) - returns
Return code.

VioWrtCharStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtCharStr - Parameters

CharStr (PCH) - input
String to be written.

Address of the character string to be written.

Length (ULONG) - input
Length of character string.

Length, in bytes, of the character string.

Row (ULONG) - input
Starting row position.

Column (ULONG) - input
Starting column position.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioWrtCharStr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtCharStr - Remarks

If a string write gets to the end of the line and is not complete, it continues at the beginning of the next line. If the write gets to the end of the screen, it terminates.

VioWrtCharStr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioWrtCharStrAtt

VioWrtCharStrAtt - Syntax

VioWrtCharStrAtt writes a character string with repeated attributes to the display.

```
#define INCL_VIO
#include <os2.h>

PCH      CharStr;    /* String to be written. */
ULONG    Length;     /* Length, in bytes, of the character string. */
ULONG    Row;        /* Starting row position. */
ULONG    Column;     /* Starting column position. */
PBYTE    Attr;       /* Attribute to be replicated. */
HVIO     VioHandle;  /* VIO presentation-space handle. */
APIRET   rc;         /* Return code. */

rc = VioWrtCharStrAtt(CharStr, Length, Row,
                      Column, Attr, VioHandle);
```

VioWrtCharStrAtt Parameter - CharStr

CharStr (PCH) - input
String to be written.

Address of the character string to be written.

VioWrtCharStrAtt Parameter - Length

Length (ULONG) - input
Length, in bytes, of the character string.

VioWrtCharStrAtt Parameter - Row

Row (ULONG) - input
Starting row position.

VioWrtCharStrAtt Parameter - Column

Column (ULONG) - input
Starting column position.

VioWrtCharStrAtt Parameter - Attr

Attr (PBYTE) - input
Attribute to be replicated.

Address of the attributes (1 or 3 bytes) to be used in the display buffer for each character of the string written.

VioWrtCharStrAtt Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioWrtCharStrAtt Return Value - rc

rc (APIRET) - returns
Return code.

VioWrtCharStrAtt returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtCharStrAtt - Parameters

CharStr (PCH) - input
String to be written.

Address of the character string to be written.

Length (ULONG) - input
Length, in bytes, of the character string.

Row (ULONG) - input
Starting row position.

Column (ULONG) - input
Starting column position.

Attr (PBYTE) - input
Attribute to be replicated.

Address of the attributes (1 or 3 bytes) to be used in the display buffer for each character of the string written.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioWrtCharStrAtt returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtCharStrAtt - Remarks

If a string write gets to the end of the line and is not complete, it continues at the beginning of the next line. If the write gets to the end of the screen, it terminates.

VioWrtCharStrAtt - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioWrtNAttr

VioWrtNAttr - Syntax

VioWrtNAttr writes an attribute to the display a specified number of times.

```
#define INCL_VIO
#include <os2.h>

PBYTE    Attr;        /* Attribute to be written. */
ULONG    Times;       /* Repeat count. */
ULONG    Row;         /* Starting row position. */
ULONG    Column;       /* Starting column position. */
HVIO     VioHandle;    /* VIO presentation-space handle. */
APIRET   rc;          /* Return code. */

rc = VioWrtNAttr(Attr, Times, Row, Column,
                VioHandle);
```

VioWrtNAttr Parameter - Attr

Attr (PBYTE) - input
Attribute to be written.

Address of the attributes (1 or 3 bytes) to be written.

VioWrtNAttr Parameter - Times

Times (ULONG) - input
Repeat count.

The number of times to write the attribute.

VioWrtNAttr Parameter - Row

Row (ULONG) - input
Starting row position.

VioWrtNAttr Parameter - Column

Column (ULONG) - input
Starting column position.

VioWrtNAttr Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioWrtNAttr Return Value - rc

rc (APIRET) - returns
Return code.

VioWrtNAttr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtNAttr - Parameters

Attr (PBYTE) - input
Attribute to be written.

Address of the attributes (1 or 3 bytes) to be written.

Times (ULONG) - input
Repeat count.

The number of times to write the attribute.

Row (ULONG) - input
Starting row position.

Column (ULONG) - input
Starting column position.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioWrtNAttr returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtNAttr - Remarks

If a repeated write gets to the end of the line and is not complete, it continues at the beginning of the next line. If the write gets to the end of the screen, it terminates.

VioWrtNAttr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioWrtNCell

VioWrtNCell - Syntax

VioWrtNCell writes a cell (character-attribute pair) to the display a specified number of times.

```
#define INCL_VIO
#include <os2.h>

PBYTE    Cell;          /* Attribute to be written. */
ULONG    Times;         /* Repeat count. */
ULONG    Row;           /* Starting row position. */
ULONG    Column;        /* Starting column position. */
HVIO     VioHandle;     /* VIO presentation-space handle. */
APIRET   rc;            /* Return code. */

rc = VioWrtNCell(Cell, Times, Row, Column,
                 VioHandle);
```

VioWrtNCell Parameter - Cell

Cell (PBYTE) - input
Attribute to be written.

The address of the character-attribute cell (2 or 4 bytes) to be written.

VioWrtNCell Parameter - Times

Times (ULONG) - input
Repeat count.

The number of times to write the attribute.

VioWrtNCell Parameter - Row

Row (ULONG) - input
Starting row position.

VioWrtNCell Parameter - Column

Column (ULONG) - input
Starting column position.

VioWrtNCell Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioWrtNCell Return Value - rc

rc (APIRET) - returns
Return code.

VioWrtNCell returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtNCell - Parameters

Cell (PBYTE) - input
Attribute to be written.

The address of the character-attribute cell (2 or 4 bytes) to be written.

Times (ULONG) - input
Repeat count.

The number of times to write the attribute.

Row (ULONG) - input
Starting row position.

Column (ULONG) - input
Starting column position.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioWrtNCell returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtNCell - Remarks

If a repeated write gets to the end of the line and is not complete, it continues at the beginning of the next line. If the write gets to the end of the screen, it terminates.

VioWrtNCell - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioWrtNChar

VioWrtNChar - Syntax

VioWrtNChar writes a character to the display a specified number of times.

```
#define INCL_VIO
#include <os2.h>

PCH      Char;      /* Character to be written. */
ULONG    Times;     /* Repeat count. */
ULONG    Row;       /* Starting row position. */
ULONG    Column;    /* Starting column position. */
HVIO     VioHandle; /* VIO presentation-space handle. */
APIRET   rc;        /* Return code. */

rc = VioWrtNChar(Char, Times, Row, Column,
                 VioHandle);
```

VioWrtNChar Parameter - Char

Char (PCH) - input
Character to be written.

The address of the character to be written.

VioWrtNChar Parameter - Times

Times (ULONG) - input
Repeat count.

The number of times to write the attribute.

VioWrtNChar Parameter - Row

Row (ULONG) - input
Starting row position.

VioWrtNChar Parameter - Column

Column (ULONG) - input
Starting column position.

VioWrtNChar Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioWrtNChar Return Value - rc

rc (APIRET) - returns
Return code.

VioWrtNChar returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtNChar - Parameters

Char (PCH) - input
Character to be written.

The address of the character to be written.

Times (ULONG) - input
Repeat count.

The number of times to write the attribute.

Row (ULONG) - input
Starting row position.

Column (ULONG) - input
Starting column position.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioWrtNChar returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtNChar - Remarks

If a repeated write gets to the end of the line and is not complete, it continues at the beginning of the next line. If the write gets to the end of the screen, it terminates.

VioWrtNChar - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VioWrtTTY

VioWrtTTY - Syntax

VioWrtTTY writes a character string to the display, starting at the current cursor position. At the completion of the write, the cursor is moved to the first position beyond the end of the string.

```
#define INCL_VIO
#include <os2.h>

PCH      CharStr;    /* String to be written. */
ULONG    Length;     /* Length of string. */
HVIO     VioHandle;  /* VIO presentation-space handle. */
APIRET   rc;         /* Return code. */

rc = VioWrtTTY(CharStr, Length, VioHandle);
```

VioWrtTTY Parameter - CharStr

CharStr (PCH) - input
String to be written.

The address of the string to be written.

VioWrtTTY Parameter - Length

Length (ULONG) - input

Length of string.

The length of the character string in bytes.

VioWrtTTY Parameter - VioHandle

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

VioWrtTTY Return Value - rc

rc (APIRET) - returns
Return code.

VioWrtTTY returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtTTY - Parameters

CharStr (PCH) - input
String to be written.

The address of the string to be written.

Length (ULONG) - input
Length of string.

The length of the character string in bytes.

VioHandle (HVIO) - input
VIO presentation-space handle.

This must be 0, unless the caller is a Presentation Manager application; in this case, it must be the value returned by [VioCreatePS](#).

rc (APIRET) - returns
Return code.

VioWrtTTY returns one of the following values:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE

VioWrtTTY - Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the screen is scrolled, and the write continues until completed.

The character's carriage return, line feed, backspace, tab, and bell are treated as commands rather than printable characters. Backspace is a nondestructive backspace. Tabs are expanded to provide standard 8-byte-wide fields. VioWrtTTY is the only video function affected by ANSI.

Characters are written using the current attribute defined by ANSI or the default value of 7.

VioWrtTTY - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

Video Data Types

This section describes the data types that are used with the Video functions, and includes the following data types:

- [FATTRS](#)
 - [FONTMETRICS](#)
 - [VIOCOLORREG](#)
 - [VIOCONFIGINFO](#)
 - [VIOCUSORINFO](#)
 - [VIOINTENSITY](#)
 - [VIOMODEINFO](#)
 - [VIOOVERSCAN](#)
 - [VIOPALSTATE](#)
 - [VIOSETTARGET](#)
 - [VIOSETLINELOC](#)
-

FATTRS

Font-attributes structure.

```
typedef struct _FATTRS {
    USHORT    usRecordLength;    /* Length of record. */
    USHORT    fsSelection;       /* Selection indicators. */
    LONG      lMatch;            /* Matched-font identity. */
    CHAR      szFacename[FACESIZE]; /* Typeface name. */
    USHORT    idRegistry;        /* Registry identifier. */
    USHORT    usCodePage;        /* Code page. */
    LONG      lMaxBaselineExt;    /* Maximum baseline extension. */
    LONG      lAveCharWidth;     /* Average character width. */
    USHORT    fsType;            /* Type indicators. */
    USHORT    fsFontUse;         /* Font-use indicators. */
} FATTRS;

typedef FATTRS *PFATTRS;
```

FATTRS Field - usRecordLength

usRecordLength (USHORT)
Length of record.

FATTRS Field - fsSelection

fsSelection (USHORT)
Selection indicators.

Flags causing the following features to be simulated by the system.

Note: If an italic flag is applied to a font that is itself defined as italic, the font is slanted further by italic simulation.

Underscore or strikeout lines are drawn using the appropriate attributes (for example, color) from the character bundle (see the CHARBUNDLE datatype), not the line bundle (see LINEBUNDLE). The width of the line, and the vertical position of the line in font space, are determined by the font. Horizontally, the line starts from a point in font space directly above or below the start point of each character, and extends to a point directly above or below the escapement point for that character.

For this purpose, the start and escapement points are those applicable to left-to-right or right-to-left character directions (see GpiSetCharDirection in *Graphics Programming Interface Programming Reference*), even if the string is currently being drawn in a top-to-bottom or bottom-to-top direction.

For left-to-right or right-to-left directions, any white space generated by the character extra and character break extra attributes (see GpiSetCharExtra and GpiSetCharBreakExtra in *Graphics Programming Interface Programming Reference*), as well as increments provided by the vector of increments on GpiCharStringPos and GpiCharStringPosAt, are also underlined/overstruck, so that in these cases the line is continuous for the string.

FATTR_SEL_ITALIC

Generate *italic* font.

FATTR_SEL_UNDERSCORE

Generate underscored font.

FATTR_SEL_BOLD

Generate **bold** font. (Note that the resulting characters are wider than those in the original font.)

FATTR_SEL_STRIKEOUT

Generate font with overstruck characters.

FATTR_SEL_OUTLINE

Use an outline font with hollow characters. If this flag is not set, outline font characters are filled. Setting this flag normally gives better performance, and for sufficiently small characters (depending on device resolution) there may be little visual difference.

FATTRS Field - lMatch

lMatch (LONG)
Matched-font identity.

FATTRS Field - szFacename[FACESIZE]

szFacename[**FACESIZE**] (CHAR)
Typeface name.

The typeface name of the font, for example, Tms Rmn.

FATTRS Field - idRegistry

idRegistry (USHORT)
Registry identifier.

Font registry identifier (zero if unknown).

FATTRS Field - usCodePage

usCodePage (USHORT)
Code page.

If zero, the current Gpi code page (see *GpiSetCp* in *Graphics Programming Interface Programming Reference*) is used. A subsequent *GpiSetCp* function changes the code page used for this logical font.

FATTRS Field - IMaxBaselineExt

IMaxBaselineExt (LONG)
Maximum baseline extension.

For raster fonts, this should be the height of the required font, in world coordinates.

For outline fonts, this should be zero.

FATTRS Field - IAveCharWidth

IAveCharWidth (LONG)
Average character width.

For raster fonts, this should be the width of the required font, in world coordinates.

For outline fonts, this should be zero.

FATTRS Field - fsType

fsType (USHORT)
Type indicators.

FATTR_TYPE_KERNING	Enable kerning (PostScript only).
FATTR_TYPE_MBCS	Font for mixed single- and double-byte code pages.
FATTR_TYPE_DBCS	Font for double-byte code pages.
FATTR_TYPE_ANTIALIASED	Antialiased font required. Only valid if supported by the device driver.

FATTRS Field - fsFontUse

fsFontUse (USHORT)
Font-use indicators.

These flags indicate how the font is to be used. They affect presentation speed and font quality.

FATTR_FONTUSE_NOMIX	Text is not mixed with graphics and can be written without regard to any interaction with graphics objects.
FATTR_FONTUSE_OUTLINE	Select an outline (vector) font. The font characters can be used as part of a path definition. If this flag is not set, an outline font might or might not be selected. If an outline font is selected, however, character widths are rounded to an integral number of pels.
FATTR_FONTUSE_TRANSFORMABLE	Characters can be transformed (for example, scaled, rotated, or sheared).

FONTMETRICS

Font-metrics structure.

This structure is returned to applications on the GpiQueryFonts and GpiQueryFontMetrics calls and conveys information from the font creator to the application.

```
typedef struct _FONTMETRICS {
    CHAR        szFamilyname[FACESIZE]; /* Family name. */
    CHAR        szFacename[FACESIZE];   /* Face name. */
    USHORT      idRegistry;              /* Registry identifier. */
    USHORT      usCodePage;              /* Code page. */
    LONG        lEmHeight;               /* Em height. */
    LONG        lXHeight;               /* X height. */
    LONG        lMaxAscender;            /* Maximum ascender. */
    LONG        lMaxDescender;          /* Maximum descender. */
    LONG        lLowerCaseAscent;        /* Lowercase ascent. */
    LONG        lLowerCaseDescent;       /* Lowercase descent. */
    LONG        lInternalLeading;         /* Internal leading. */
    LONG        lExternalLeading;         /* External leading. */
    LONG        lAveCharWidth;           /* Average character width. */
    LONG        lMaxCharInc;             /* Maximum character increment. */
    LONG        lEmInc;                 /* Em increment. */
    LONG        lMaxBaselineExt;         /* Maximum baseline extent. */
    SHORT       sCharSlope;              /* Character slope. */
    SHORT       sInlineDir;              /* Inline direction. */
    SHORT       sCharRot;                /* Character rotation. */
    USHORT      usWeightClass;           /* Weight class. */
    USHORT      usWidthClass;            /* Width class. */
}
```

```

SHORT    sXDeviceRes;        /* X-device resolution. */
SHORT    sYDeviceRes;        /* Y-device resolution. */
SHORT    sFirstChar;         /* First character. */
SHORT    sLastChar;          /* Last character. */
SHORT    sDefaultChar;        /* Default character. */
SHORT    sBreakChar;          /* Break character. */
SHORT    sNominalPointSize;    /* Nominal point size. */
SHORT    sMinimumPointSize;    /* Minimum point size. */
SHORT    sMaximumPointSize;    /* Maximum point size. */
USHORT    fsType;             /* Type indicators. */
USHORT    fsDefn;             /* Definition indicators. */
USHORT    fsSelection;        /* Selection indicators. */
USHORT    fsCapabilities;     /* Font capabilities. */
LONG      lSubscriptXSize;     /* Subscript x-size. */
LONG      lSubscriptYSize;     /* Subscript y-size. */
LONG      lSubscriptXOffset;    /* Subscript x-offset. */
LONG      lSubscriptYOffset;    /* Subscript y-offset. */
LONG      lSuperscriptXSize;    /* Superscript x-size. */
LONG      lSuperscriptYSize;    /* Superscript y-size. */
LONG      lSuperscriptXOffset;  /* Superscript x-offset. */
LONG      lSuperscriptYOffset;  /* Superscript y-offset. */
LONG      lUnderscoreSize;      /* Underscore size. */
LONG      lUnderscorePosition;  /* Underscore position. */
LONG      lStrikeoutSize;       /* Strikeout size. */
LONG      lStrikeoutPosition;   /* Strikeout position. */
SHORT    sKerningPairs;        /* Kerning pairs. */
SHORT    sFamilyClass;         /* Font family design classification. */
LONG      lMatch;              /* Matched font identity. */
PANOSE    panose;              /* Panose font descriptor. */
} FONTMETRICS;

typedef FONTMETRICS *PFONTMETRICS;

```

FONTMETRICS Field - szFamilyname[FACESIZE]

szFamilyname[FACESIZE] (CHAR)
Family name.

The family name of the font that describes the basic appearance of the font, for example, Times New Roman** This string is null terminated, and therefore is limited to 31 characters in length.

FONTMETRICS Field - szFacename[FACESIZE]

szFacename[FACESIZE] (CHAR)
Face name.

The typeface name that defines the particular font, for example, Times New Roman Bold Italic. This string is null terminated, and therefore is limited to 31 characters in length.

FONTMETRICS Field - idRegistry

idRegistry (USHORT)
Registry identifier.

The IBM registered number (or zero).

FONTMETRICS Field - usCodePage

usCodePage (USHORT)
Code page.

Defines the registered code page supported by the font. For example, the original IBM PC code page is 437. A value of 0 implies that the font may be used with any of the OS/2 supported code pages.

Where a font contains special symbols for which there is no registered code page, then code page 65400 is used.

FONTMETRICS Field - IEmHeight

IEmHeight (LONG)
Em height.

The height of the Em square in world coordinate units. This corresponds to the point size for the font.

FONTMETRICS Field - IXHeight

IXHeight (LONG)
X height.

The nominal height above the baseline for lowercase characters (ignoring ascenders) in world coordinate units.

FONTMETRICS Field - IMaxAscender

IMaxAscender (LONG)
Maximum ascender.

The maximum height above the baseline reached by any part of any symbol in the font in world coordinate units. This field may exceed *IEmHeight*.

FONTMETRICS Field - IMaxDescender

IMaxDescender (LONG)
Maximum descender.

The maximum depth below the baseline reached by any part of any symbol in the font in world coordinate units. This field may exceed *IEmHeight*.

FONTMETRICS Field - ILowerCaseAscent

ILowerCaseAscent (LONG)

Lowercase ascent.

The maximum height above the baseline reached by any part of any lowercase (Latin unaccented "a" through "z") symbol in the font in world coordinate units.

FONTMETRICS Field - ILowerCaseDescent

ILowerCaseDescent (LONG)

Lowercase descent.

The maximum depth below the baseline reached by any part of any lowercase (Latin unaccented "a" through "z") symbol in the font in world coordinate units.

FONTMETRICS Field - IInternalLeading

IInternalLeading (LONG)

Internal leading.

The amount of space which, when subtracted from *IMaxAscender*, gives a font-design dependent, but glyph-set independent, measure of the distance above the baseline that characters extend. This calculation approximates the visual top to a row of characters without actually looking at the characters in the row.

For optimum results, this field should be used by applications to position the first line of a block of text by subtracting it from *IMaxAscender* and positioning the baseline that distance below whatever is above the text.

Note: This does not guarantee that characters will not overwrite information above them, but does give a font designer's view of where to place the text. Collision should be tested for, and additional space allocated if necessary.

FONTMETRICS Field - IExternalLeading

IExternalLeading (LONG)

External leading.

The amount of guaranteed white space advised by the font designer to appear between adjacent rows of text. This value may be zero.

Note: The fonts built in to Presentation Manager have zero in this field.

FONTMETRICS Field - IAveCharWidth

IAveCharWidth (LONG)
Average character width.

This is determined by multiplying the width of each lowercase character by a constant, adding the products, and then dividing by 1000.
The letters involved in this, plus their constants, are as follows:

Letter	Constant
a	64
b	14
c	27
d	35
e	100
f	20
g	14
h	42
i	63
j	3
k	6
l	35
m	20
n	56
o	56
p	17
q	4
r	49
s	56
t	71
u	31
v	10
w	18
x	3
y	18
z	2
space	166

Note: For fixed pitch fonts, this value will be the same as the (A width + B width + C width) escapement of each character.

FONTMETRICS Field - IMaxCharInc

IMaxCharInc (LONG)
Maximum character increment.

The maximum character increment for the font in world coordinate units.

FONTMETRICS Field - IEmlInc

IEmlInc (LONG)
Em increment.

The width of the Em square in world coordinate units. This corresponds to the point size of the font. When the horizontal device resolution equals the vertical device resolution this is equal to the em height.

FONTMETRICS Field - IMaxBaselineExt

IMaxBaselineExt (LONG)

Maximum baseline extent.

The maximum vertical space occupied by the font, in world coordinate units. This is the sum of *IMaxAscender* and *IMaxDescender* if both are positive. It is also the sum of *lInternalLeading* and *lEmHeight* .

One possible type of line spacing can be computed by adding *IMaxBaselineExt* to *lExternalLeading* . Such a line spacing, however, would be dependent on the glyph set included in the font. If a new version of the font should be made available, with new glyphs, then it is possible that this value will change because one of the new glyphs has gone above the previous *IMaxAscender* or below the previous *IMaxDescender* . More sophisticated applications will base line spacing on the point size (*lEmHeight*) of the font, which is an invariant of the font, multiplied by some factor (for example, 120%) plus any external leading.

This field may exceed *lEmHeight* .

FONTMETRICS Field - sCharSlope

sCharSlope (SHORT)

Character slope.

Defines the nominal slope for the characters of a font. The slope is defined in degrees increasing clockwise from the vertical. An *italic* font is an example of a font with a nonzero slope.

Note: The units for this metric are degrees and minutes, encoded as shown in the following example:

180 degrees 59 minutes would be represented as :

< byte 1 >	< byte 2 >
< Minutes >	< Degrees >
0 1 1 1 0 1 1	0 1 0 1 1 0 1 0 0
59 min	180 degrees

FONTMETRICS Field - sInlineDir

sInlineDir (SHORT)

Inline direction.

The direction in which the characters in the font are designed for viewing. The direction is defined in degrees increasing clockwise from the horizontal (left-to-right). Characters are added to a line of text in the inline direction.

Note: The units for this metric are degrees and minutes, encoded as shown in *sCharSlope* .

FONTMETRICS Field - sCharRot

sCharRot (SHORT)

Character rotation.

The rotation of the character glyphs with respect to the baseline, the angle increasing counter clockwise. This is the angle assigned by the font designer.

Note: The units for this metric are degrees and minutes, encoded as shown in *sCharSlope* .

FONTMETRICS Field - usWeightClass

usWeightClass (USHORT)

Weight class.

Indicates the visual weight (thickness of strokes) of the characters in the font:

	Value	Description
1		Ultra-light
2		Extra-light
3		Light
4		Semi-light
5		Medium (normal)
6		Semi-bold
7		Bold
8		Extra-bold
9		Ultra-bold

FONTMETRICS Field - usWidthClass

usWidthClass (USHORT)

Width class.

Indicates the relative aspect ratio of the characters of the font in relation to the normal aspect ratio for this type of font:

Value	Description	% of normal width
1	Ultra-condensed	50
2	Extra-condensed	62.5
3	Condensed	75
4	Semi-condensed	87.5
5	Medium (normal)	100
6	Semi-expanded	112.5
7	Expanded	125
8	Extra-expanded	150
9	Ultra-expanded	200

FONTMETRICS Field - sXDeviceRes

sXDeviceRes (SHORT)

X-device resolution.

For bit-map fonts this is the resolution in the X direction of the intended target device, measured in pels per inch.

For outline fonts this is the number of notional units in the X direction of the Em square, measured in notional units per Em. (Notional units are the units in which the outline is defined.)

FONTMETRICS Field - sYDeviceRes

sYDeviceRes (SHORT)
Y-device resolution.

For bit-map fonts this is the resolution in the Y direction of the intended target device, measured in pels per inch.

For outline fonts this is the number of notional units in the Y direction of the Em square, measured in notional units per Em. (Notional units are the units in which the outline is defined.)

FONTMETRICS Field - sFirstChar

sFirstChar (SHORT)
First character.

The code point of the first character in the font.

FONTMETRICS Field - sLastChar

sLastChar (SHORT)
Last character.

The code point of the last character in the font, expressed as an offset from *sFirstChar* .

All code points between the first and last character specified must be supported by the font.

FONTMETRICS Field - sDefaultChar

sDefaultChar (SHORT)
Default character.

The code point that is used if a code point outside the range supported by the font is used, expressed as an offset from *sFirstChar* .

FONTMETRICS Field - sBreakChar

sBreakChar (SHORT)
Break character.

The code point that represents the "space" or "break" character for this font, expressed as an offset from *sFirstChar* . For example, if the first character is the space in code page 850, *sFirstChar* = 32, and *sBreakChar* = 0.

FONTMETRICS Field - sNominalPointSize

sNominalPointSize (SHORT)
Nominal point size.

For a bit-map font, this field contains the height of the font.

For an outline font, this field contains the height intended by the font designer. For example, some fonts are designed for text use in which case a value of 120 (12 point) would probably be placed in this field, whereas other fonts are designed for "display" use ("display" is typographer's terminology for larger sizes). This is not the only size at which the font can be used.

Measured in decipoints (a decipoint is 1/720th of an inch).

FONTMETRICS Field - sMinimumPointSize

sMinimumPointSize (SHORT)
Minimum point size.

For a bit-map font, this field does not apply. For an outline font, this field contains the minimum height intended by the font designer. Note that this is not a restriction of the size at which the font can be used.

Measured in decipoints (a decipoint is 1/720th of an inch).

FONTMETRICS Field - sMaximumPointSize

sMaximumPointSize (SHORT)
Maximum point size.

For a bit-map font, this field does not apply.

For an outline font, this field contains the maximum height intended by the font designer. Note that this is not a restriction of the size at which the font can be used.

Measured in decipoints (a decipoint is 1/720th of an inch).

FONTMETRICS Field - fsType

fsType (USHORT)
Type indicators.

This field contains the following information:

FM_TYPE_FIXED	Characters in the font have the same fixed width.
FM_TYPE_LICENSED	Licensed (protected) font.
FM_TYPE_KERNING	Font contains kerning information.
FM_TYPE_64K	Font is larger than 64KB (KB equals 1024 bytes) in size. If the following two bits are false, the font is for single-byte code pages. One of the bits may be set.
FM_TYPE_DBCS	Font is for double-byte code pages.
FM_TYPE_MBCS	Font is for mixed single- or double-byte code pages.

FONTMETRICS Field - fsDefn

fsDefn (USHORT)

Definition indicators.

Contains the following font definition data:

FM_DEFN_OUTLINE	Font is a vector (outline) font; otherwise, it is a bit-map font.
FM_DEFN_GENERIC	Font is in a format that can be used by the GPI; otherwise, it is a device font.

FONTMETRICS Field - fsSelection

fsSelection (USHORT)

Selection indicators.

Contains information about the font patterns in the physical font.

Note: The flags do not reflect simulations applied to the physical font.

Possible values are:

FM_SEL_ITALIC	True indicates that this font is designed as an italic font.
FM_SEL_UNDERSCORE	TRUE indicates that this font is designed with underscores included in each character.
FM_SEL_NEGATIVE	TRUE indicates that this font is designed with the background and foreground reversed.
FM_SEL_OUTLINE	TRUE indicates that this font is designed with outline (hollow) characters.
FM_SEL_STRIKEOUT	TRUE indicates that this font is designed with an overstrike through each character.
FM_SEL_BOLD	TRUE indicates that this font is designed with bold characters.
FM_SEL_ISO9241_TESTED	This flag indicates that the font has been tested for compliance to ISO 9241. The presence of this flag doesn't indicate whether the font passed or failed, only that it was tested.

Note: While the fonts were primarily tested for meeting the ISO standard, they have also been designed to meet the German standard DIN 66 234. Where the two standards differ, the fonts have been designed to meet the more

FONTMETRICS Field - fsCapabilities

fsCapabilities (USHORT)
Font capabilities.

This attribute applies only to device fonts.

FM_CAP_NOMIX

Characters may not be mixed with graphics.

QUALITY

The most significant byte may contain the following numeric value:

0	Undefined
1	DP quality
2	DP draft
3	Near Letter Quality
4	Letter Quality

FONTMETRICS Field - ISubscriptXSize

ISubscriptXSize (LONG)
Subscript x-size.

The horizontal size recommended by the font designer for subscripts for this font in world coordinate units.

FONTMETRICS Field - ISubscriptYSize

ISubscriptYSize (LONG)
Subscript y-size.

The vertical size recommended by the font designer for subscripts for this font in world coordinate units.

FONTMETRICS Field - ISubscriptXOffset

ISubscriptXOffset (LONG)
Subscript x-offset.

The baseline x-offset recommended by the font designer for subscripts for this font in world coordinate units.

FONTMETRICS Field - ISubscriptYOffset

ISubscriptYOffset (LONG)

Subscript y-offset.

The baseline y-offset recommended by the font designer for subscripts for this font in world coordinate units.

Note: Positive numbers indicate an offset below the baseline.

FONTMETRICS Field - ISuperscriptXSize

ISuperscriptXSize (LONG)

Superscript x-size.

The horizontal size recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - ISuperscriptYSize

ISuperscriptYSize (LONG)

Superscript y-size.

The vertical point size recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - ISuperscriptXOffset

ISuperscriptXOffset (LONG)

Superscript x-offset.

The baseline x-offset recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - ISuperscriptYOffset

ISuperscriptYOffset (LONG)

Superscript y-offset.

The baseline y-offset recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - IUnderscoreSize

IUnderscoreSize (LONG)

Underscore size.

The width (thickness) of the underscore stroke in world coordinate units. This describes the actual underscore in the font if FM_SEL_UNDERSCORE is also set. Otherwise it describes what the engine will simulate if underscore is requested in GpiCreateLogFont.

FONTMETRICS Field - IUnderscorePosition

IUnderscorePosition (LONG)

Underscore position.

The position of the underscore stroke from the baseline in world coordinate units. This describes the actual underscore in the font if FM_SEL_UNDERSCORE is also set. Otherwise it describes what the engine will simulate if underscore is requested in GpiCreateLogFont.

Note: Positive values indicate an offset below the baseline.

FONTMETRICS Field - IStrikeoutSize

IStrikeoutSize (LONG)

Strikeout size.

The width of the strikeout stroke in world coordinate units. This describes the actual underscore in the font if FM_SEL_STRIKEOUT is also set. Otherwise it describes what the engine will simulate if overstrike is requested in GpiCreateLogFont.

FONTMETRICS Field - IStrikeoutPosition

IStrikeoutPosition (LONG)

Strikeout position.

The position of the strikeout stroke relative to the baseline in world coordinate units. This describes the actual underscore in the font if FM_SEL_STRIKEOUT is also set. Otherwise it describes what the engine will simulate if overstrike is requested in GpiCreateLogFont.

FONTMETRICS Field - sKerningPairs

sKerningPairs (SHORT)

Kerning pairs.

The number of kerning pairs in the kerning pair table.

FONTMETRICS Field - sFamilyClass

sFamilyClass (SHORT)

Font family design classification.

This value contains a font class and its subclass.

FONTMETRICS Field - IMatch

IMatch (LONG)

Matched font identity.

This uniquely identifies the font for a given device and device driver combination. A positive match number signifies that the font is a generic (engine) font while a negative number indicates a device font (a native or downloadable font). This value should not be used to identify a font across system boundaries.

FONTMETRICS Field - panose

panose (PANOSE)

Panose font descriptor.

This is the Panose descriptor identifying the visual characteristics of the font.

VIOCOLORREG

VIOCOLORREG data structure.

```
typedef struct _VIOCOLORREG {
    USHORT    cb;           /* Length of this data structure in bytes. */
    USHORT    type;         /* Request type = 3, get color registers. */
    USHORT    firstcolorreg; /* First color register. */
    USHORT    numcolorregs; /* Number of color registers to get. */
    PCH       colorregaddr; /* Area where the color registers are returned. */
} VIOCOLORREG;
```

```
typedef VIOCOLORREG *PVIOCOLORREG;
```

VIOCOLORREG Field - cb

cb (USHORT)

Length of this data structure in bytes.

Length of the structure, including length. The maximum valid value is 12.

VIOCOLORREG Field - type

type (USHORT)

Request type = 3, get color registers.

Request type 3 for color registers.

VIOCOLORREG Field - firstcolorreg

firstcolorreg (USHORT)

First color register.

The first color register to get in the color register sequence. It must be specified in the range 0 through 255. The color registers are returned in sequential order.

VIOCOLORREG Field - numcolorregs

numcolorregs (USHORT)

Number of color registers to get.

The number of color registers to get; must be specified in the range 1 through 256.

VIOCOLORREG Field - colorregaddr

colorregaddr (PCH)

Area where the color registers are returned.

The far address of a data area where the color registers are returned. The size of the data area must be three bytes times the number of color registers to get. The format of each entry returned is as follows:

Byte 1	Red value
Byte 2	Green value
Byte 3	Blue value

VIOCONFIGINFO

VIOCONFIGINFO data structure.

```
typedef struct _VIOCONFIGINFO {
```

```

ULONG      cb;                /* Length of this data structure */
ULONG      adapter;           /* Display adapter type */
ULONG      display;           /* Display monitor type */
ULONG      cbMemory;          /* Bytes of memory on the adapter */
ULONG      Configuration;     /* Configuration number */
ULONG      VDHVersion;        /* Reserved */
ULONG      Flags;
ULONG      HWBufferSize;      /* Size to save video state */
ULONG      FullSaveSize;      /* Full save size. */
ULONG      PartSaveSize;      /* Partial save size */
ULONG      EMAdaptersOff;     /* Offset to emulated adapter types */
ULONG      EMDisplayOFF;      /* Offset to emulated display types */
} VIOCONFIGINFO;

typedef VIOCONFIGINFO *PVIOCONFIGINFO;

```

VIOCONFIGINFO Field - cb

cb (ULONG)
Length of this data structure

Input parameter to VioGetConfig. The length must be either 4, to indicate the actual length should be returned, or at least 48.

VIOCONFIGINFO Field - adapter

adapter (ULONG)
Display adapter type

Display adapter type.

	Value	Definition
0-3		Reserved
3		VGA Display Adapter
4-6		Reserved
7		8514/A
8		Image Adapter/A
9		XGA Display Adapter

Values ranging from 0-4095 are reserved for IBM.

VIOCONFIGINFO Field - display

display (ULONG)
Display monitor type

Display or monitor type.

	Value	Definition
0-2		Reserved
3		13-inch Monochrome Display
4		13-inch Color Display
5-8		Reserved

9	16-inch 1024x768 capable color display
10	LCD or Plasma display
11	Large monochrome display
12	14-inch 1024x768 capable color display
13	Reserved

Values ranging from 0-4095 are reserved for IBM.

VIOCONFIGINFO Field - cbMemory

cbMemory (ULONG)

Bytes of memory on the adapter

Amount of memory, in bytes, on the adapter.

VIOCONFIGINFO Field - Configuration

Configuration (ULONG)

Configuration number

Number of the display configuration that this data corresponds to. This is assigned by the video subsystem.

VIOCONFIGINFO Field - VDHVersion

VDHVersion (ULONG)

Reserved

This field is reserved.

VIOCONFIGINFO Field - Flags

Flags (ULONG)

The flag bits are defined as follows:

	Bit	Description
31-1		Reserved
0		Power-up display configuration.

VIOCONFIGINFO Field - HWBufferSize

HWBufferSize (ULONG)
Size to save video state

Size of the buffer required by the Base Video Handler (BVH) to save the full hardware state, excluding the physical display buffer.

VIOCONFIGINFO Field - FullSaveSize

FullSaveSize (ULONG)
Full save size.

Maximum size buffer required by the BVH to save the full physical display buffer.

VIOCONFIGINFO Field - PartSaveSize

PartSaveSize (ULONG)
Partial save size

Maximum size buffer required by the BVH to save the portion of the physical display buffer that is overlaid by a pop-up.

VIOCONFIGINFO Field - EMAdaptersOff

EMAdaptersOff (ULONG)
Offset to emulated adapter types

Offset, within the configuration data structure, to the following information describing what other display adapters are emulated by this display adapter.

Number of Data words (ULONG)

Contains a one-word field, specifying a count of data words to follow.

Data word 1 (ULONG)

Bits, set in the data words, identify display adapters emulated. Data word 1 has the following definition:

Bit	Description
0-2	Reserved
3	VGA
4-6	Reserved
7	8514/A Adapter
8	Image Adapter/A
9	XGA Adapter
10-31	Reserved

VIOCONFIGINFO Field - EMDisplayOFF

EMDisplayOFF (ULONG)

Offset to emulated display types

Offset, within the configuration data structure, to the following information describing what other displays are emulated by this display.

Number of Data words (ULONG)

One-word field, specifying a count of data words to follow.

Data word 1 (ULONG)

Bits, set in the data words, identify displays emulated. Data word 1 has the following definition:

Bit	Description
0-2	Reserved
3	13-inch Monochrome Display
4	13-inch Color Display
5-8	Reserved
9	16-inch 1024x768 capable color display
10	LCD or Plasma display
11	Large monochrome display
12	14-inch 1024x768 capable color display
13-31	Reserved

VIOCURSORINFO

VIOCURSORINFO data structure.

```
typedef struct _VIOCURSORINFO {
    USHORT    yStart; /* Cursor start line. */
    USHORT    cEnd;   /* Cursor end line. */
    USHORT    cx;     /* Cursor width. */
    USHORT    attr;    /* -1 =hidden cursor. */
} VIOCURSORINFO;
```

```
typedef VIOCURSORINFO *PVIOCURSORINFO;
```

VIOCURSORINFO Field - yStart

yStart (USHORT)

Cursor start line.

Horizontal scan line in the character cell that marks the top line of the cursor. If the character cell has n scan lines, 0 is the top scan line of the character cell and (n-1) is the bottom scan line.

VIOCURSORINFO Field - cEnd

cEnd (USHORT)

Cursor end line.

Horizontal scan line in the character cell that marks the bottom line of the cursor. Scan lines within a character cell are numbered as

defined in startline.

VIOCURSORINFO Field - cx

cx (USHORT)
Cursor width.

Width of the cursor. In text modes, cursorwidth is the number of columns. The maximum number supported by the OS/2 base video subsystem is 1. In graphics mode, cursorwidth is the number of pels.

VIOCURSORINFO Field - attr

attr (USHORT)
-1 =hidden cursor.

A value of -1 denotes a hidden cursor; all other values in text mode denote normal cursor and in graphics mode denote color attribute.

VIOINTENSITY

VIOINTENSITY data structure.

```
typedef struct _VIOINTENSITY {
    USHORT    cb;      /* Length of this data structure in bytes. */
    USHORT    type;    /* Request type = 2, get blink/background intensity switch. */
    USHORT    fs;      /* Value of blink/background switch. */
} VIOINTENSITY;

typedef VIOINTENSITY *PVIOINTENSITY;
```

VIOINTENSITY Field - cb

cb (USHORT)
Length of this data structure in bytes.

Length of the structure, including length. The only valid value is 6.

VIOINTENSITY Field - type

type (USHORT)
Request type = 2, get blink/background intensity switch.

VIOINTENSITY Field - fs

fs (USHORT)

Value of blink/background switch.

Switch set as:

	Value	Definition
0		Blinking foreground colors enabled.
1		High-intensity background colors enabled.

VIOMODEINFO

VIOMODEINFO data structure.

```
typedef struct _VIOMODEINFO {
    USHORT    cb;                /* Length of this data structure */
    UCHAR     fbType;            /* Bit mask of mode being set. */
    UCHAR     color;             /* Number of colors (power of 2). */
    USHORT    col;               /* The number of text columns. */
    USHORT    row;               /* The number of text rows. */
    USHORT    hres;              /* Horizontal resolution. */
    USHORT    vres;              /* Vertical resolution. */
    UCHAR     fmt_ID;            /* Attribute format. */
    UCHAR     attrib;            /* Number of attributes. */
    USHORT    resv;              /* Reserved. */
    ULONG     buf_addr;          /* Video aperture address. */
    ULONG     buf_length;        /* Video aperture length. */
    ULONG     full_length;       /* Video state full save length. */
    ULONG     partial_length;    /* Video state partial save length. */
    ULONG     ext_data_addr;     /* Extra data address. */
} VIOMODEINFO;

typedef VIOMODEINFO *PVIOMODEINFO;
```

VIOMODEINFO Field - cb

cb (USHORT)

Length of this data structure

Input parameter to VioGetMode. Length specifies the length of the data structure in bytes, including length. The value specified on input controls the amount of mode data returned. The minimum structure size required is two bytes. The length is modified on output.

VIOMODEINFO Field - fbType

fbType (UCHAR)

Bit mask of mode being set.

Mode characteristics bit mask:

	Bit	Description
7-4		Reserved
3		0 = VGA BIOS compatible modes 1 = Native mode
2		0 = Enable color burst 1 = Disable color burst
1		0 = Text mode 1 = Graphics mode
0		0 = Monochrome compatible mode 1 = Other

VIOMODEINFO Field - color

color (UCHAR)

Number of colors (power of 2).

Number of colors defined as a power of 2. This is equivalent to the number of color bits that define the color, as in this example:

	Value	Definition
0		Monochrome
1		2 colors
2		4 colors
4		16 colors
8		256 colors
16		64K colors
24		16M colors

VIOMODEINFO Field - col

col (USHORT)

The number of text columns.

The number of text columns.

VIOMODEINFO Field - row

row (USHORT)

The number of text rows.

The number of text rows.

VIOMODEINFO Field - hres

hres (USHORT)

Horizontal resolution.

Horizontal resolution; the number of pel columns.

VIOMODEINFO Field - vres

vres (USHORT)
Vertical resolution.

Vertical resolution, the number of pel rows.

VIOMODEINFO Field - fmt_ID

fmt_ID (UCHAR)
Attribute format.

The format of the attributes.

VIOMODEINFO Field - attrib

attrib (UCHAR)
Number of attributes.

The number of attributes in a character cell.

VIOMODEINFO Field - resv

resv (USHORT)
Reserved.

VIOMODEINFO Field - buf_addr

buf_addr (ULONG)
Video aperture address.

The physical address of the physical display aperture. This may be zero for emulated video hardware.

VIOMODEINFO Field - buf_length

buf_length (ULONG)
Video aperture length.

The length of the physical display aperture.

VIOMODEINFO Field - full_length

full_length (ULONG)
Video state full save length.

The size of the buffer required for a full save of the video state.

VIOMODEINFO Field - partial_length

partial_length (ULONG)
Video state partial save length.

The size of the buffer required for a partial (pop-up) save of the video state.

VIOMODEINFO Field - ext_data_addr

ext_data_addr (ULONG)
Extra data address.

The virtual address of an extended mode data structure or zero (if none).. The format of the extended mode data structure is determined by the device driver and is unknown to OS/2.

VIOOVERSCAN

VIOOVERSCAN data structure.

```
typedef struct _VIOOVERSCAN {
    USHORT    cb;        /* Length of this data structure in bytes. */
    USHORT    type;      /* Request type = 1, get overscan. */
    USHORT    color;     /* Color value. */
} VIOOVERSCAN;

typedef VIOOVERSCAN *PVIOOVERSCAN;
```

VIOOVERSCAN Field - cb

cb (USHORT)

Length of this data structure in bytes.

Length of the structure, including length. The only valid value is 6.

VIOOVERSCAN Field - type

type (USHORT)

Request type = 1, get overscan.

Request type 1 for overscan (border) color.

VIOOVERSCAN Field - color

color (USHORT)

Color value.

The color value.

VIOPALSTATE

VIOPALSTATE data structure.

```
typedef struct _VIOPALSTATE {
    USHORT    cb;           /* Length of this data structure in bytes. */
    USHORT    type;         /* Request type = 0, get palette registers. */
    USHORT    iFirst;       /* First palette register to return. */
    USHORT    acolor[1];    /* Color value palette register. */
} VIOPALSTATE;

typedef VIOPALSTATE *PVIOPALSTATE;
```

VIOPALSTATE Field - cb

cb (USHORT)
Length of this data structure in bytes.

Length of the structure, including length. The maximum valid value is 38.

VIOPALSTATE Field - type

type (USHORT)
Request type = 0, get palette registers.

Request type 0 for palette registers.

VIOPALSTATE Field - iFirst

iFirst (USHORT)
First palette register to return.

The first palette register in the palette register sequence. It must be specified in the range 0 through 15. The palette registers are returned in sequential order. The number returned is based upon length.

VIOPALSTATE Field - acolor[1]

acolor[1] (USHORT)
Color value palette register.

The color value for each palette register. The maximum number of entries in the color value array is 16.

VIOSETTARGET

VIOSETTARGET data structure.

```
typedef struct _VIOSETTARGET {  
    USHORT    cb;           /* Length of this data structure in bytes. */  
    USHORT    type;         /* Request type = 6, get display configuration. */  
    USHORT    defaultalgorithm;  
} VIOSETTARGET;
```

```
typedef VIOSETTARGET *PVIOSETTARGET;
```

VIOSETTARGET Field - cb

cb (USHORT)

Length of this data structure in bytes.

Length of the structure, including length. The only valid value is 6.

VIOSETTARGET Field - type

type (USHORT)

Request type = 6, get display configuration.

Request type 6 to get display configuration selected to be the target of the next [VioSetMode](#).

VIOSETTARGET Field - defaultalgorithm

defaultalgorithm (USHORT)

Configuration:

	Value	Definition
0		Default selection algorithm. See VioSetMode .
1		Primary
2		Secondary

VIOSETULINELOC

VIOSETULINELOC data structure.

```
typedef struct _VIOSETULINELOC {
    USHORT    cb;          /* Length of this data structure in bytes. */
    USHORT    type;        /* Request type = 5, get scan line for underlining. */
    USHORT    scanline;
} VIOSETULINELOC;

typedef VIOSETULINELOC *PVIOSETULINELOC;
```

VIOSETULINELOC Field - cb

cb (USHORT)

Length of this data structure in bytes.

Length of the structure, including length. The only valid value is 6.

VIOSETULINELOC Field - type

type (USHORT)

Request type = 5, get scan line for underlining.

Request type 5 to get the scan line for underlining.

VIOSETULINELOC Field - scanline

scanline (USHORT)

The value returned is in the range 0 through 31 and is the scan line minus 1. A value of 32 means underlining is disabled.

Presentation Manager

This chapter is an addendum to the *Presentation Manager Programming Reference* .

The chapter includes the following sections:

- [Debug Functions](#)
 - [Graphics Functions](#)
 - [Hook Functions](#)
 - [Window Functions](#)
 - [PM Data Types](#)
-

Debug Functions

This section describes the debug functions that are new or enhanced on OS/2 Warp (PowerPC Edition). The debug functions described in this section are:

- [DebugOutput](#)
-

DebugOutput

DebugOutput - Syntax

DebugOutput sends a string to the debug terminal.

```
#define INCL_PM
#include <os2.h>

PSZ      string;
APIRET   rc;

rc = DebugOutput(string);
```

DebugOutput Parameter - string

string (PSZ) - input
A pointer to a character string.

DebugOutput Return Value - rc

rc (APIRET) - returns
Return values.

TRUE	The string was written out successfully.
FALSE	The string was not written out.

DebugOutput - Parameters

string (PSZ) - input
A pointer to a character string.

rc (APIRET) - returns
Return values.

TRUE	The string was written out successfully.
FALSE	The string was not written out.

DebugOutput - Remarks

Strings are only output if the environment variable PM_DEBUGOUTPUT is set to ON in the CONFIG.SYS file.

DebugOutput - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

Graphics Functions

This section describes the Graphics Programming Interface (GPI) functions that are new or enhanced on OS/2 Warp (PowerPC Edition). The GPI functions described in this section are:

- [GpiSuspendPlay](#)
- [GpiResumePlay](#)

GpiResumePlay

GpiResumePlay - Syntax

GpiResumePlay resumes the playing of a metafile.

```
#include <os2.h>

HPS      hps;
BOOL     Success;
APIRET   rc;    /* Success indicator. */

rc = GpiResumePlay(hps, Success);
```

GpiResumePlay Parameter - hps

hps (HPS) - input
Presentation-space handle.

GpiResumePlay Parameter - Success

Success (BOOL) - input
Success indicator.

GpiResumePlay Return Value - rc

rc (APIRET) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

GpiResumePlay - Parameters

hps (HPS) - input
Presentation-space handle.

Success (BOOL) - input
Success indicator.

rc (APIRET) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

GpiResumePlay - Remarks

Use GpiResumePlay after issuing [GpiSuspendPlay](#) to resume the playing of the metafile. This function can be used only with a presentation space of type GPIT_NORMAL.

GpiResumePlay - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

GpiSuspendPlay

GpiSuspendPlay - Syntax

GpiSuspendPlay suspends the playing of a metafile.

```
#include <os2.h>

HPS      hps;
BOOL     Success;
APIRET   rc;    /* Success indicator. */

rc = GpiSuspendPlay(hps, Success);
```

GpiSuspendPlay Parameter - hps

hps (HPS) - input
Presentation-space handle.

GpiSuspendPlay Parameter - Success

Success (BOOL) - input
Success indicator.

GpiSuspendPlay Return Value - rc

rc (APIRET) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

GpiSuspendPlay - Parameters

hps (HPS) - input
Presentation-space handle.

Success (BOOL) - input
Success indicator.

rc (APIRET) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

GpiSuspendPlay - Remarks

GpiSuspendPlay suspends the play of a metafile that is in progress on another thread to the same presentation space. It is used by a queue processor to suspend the processing of a print job. The play is suspended at some convenient point. This function does not wait for play to be suspended before completion.

This function can be used only with a presentation space of type GPIT_NORMAL, which is the default setting of GpiCreatePS.

GpiSuspendPlay - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

Hook Functions

This section describes the hook functions that the application can provide that are new or enhanced on OS/2 Warp (PowerPC Edition). The hook functions described in this section are:

- [AbnormalInputTimeoutHook](#)
 - [CallHookHook](#)
-

AbnormalInputTimeoutHook

AbnormalInputTimeoutHook - Syntax

AbnormalInputTimeoutHook is called for debugging purposes, whenever the session manager detects an application that is not processing user input correctly.

```
#define INCL_xxx
#include <os2.h>

HMQ    hmqCurrent;
PID     pid;
TID     tid;
BOOL    Processed;    /* Success indicator. */
```

```
Processed = AbnormalInputTimeoutHook(hmqCurrent,  
                                     pid, tid);
```

AbnormalInputTimeoutHook Parameter - hmqCurrent

hmqCurrent ([HMQ](#)) - input
Message queue handle associated with the current thread.

AbnormalInputTimeoutHook Parameter - pid

pid ([PID](#)) - input
Process identity of the application that contains *tid* .

AbnormalInputTimeoutHook Parameter - tid

tid ([TID](#)) - input
Identity of the thread for which the input system is waiting.

AbnormalInputTimeoutHook Return Value - Processed

Processed ([BOOL](#)) - returns
Success indicator.

TRUE	The next hook in the chain is not called, or the Switch To dialog box is not displayed by the session manager.
FALSE	The next hook in the chain is called, or the Switch To dialog box is displayed by the session manager.

AbnormalInputTimeoutHook - Parameters

hmqCurrent ([HMQ](#)) - input
Message queue handle associated with the current thread.

pid ([PID](#)) - input
Process identity of the application that contains *tid* .

tid ([TID](#)) - input
Identity of the thread for which the input system is waiting.

Processed (BOOL) - returns
Success indicator.

TRUE	The next hook in the chain is not called, or the Switch To dialog box is not displayed by the session manager.
FALSE	The next hook in the chain is called, or the Switch To dialog box is displayed by the session manager.

AbnormalInputTimeoutHook - Remarks

AbnormalInputTimeoutHook allows a debug application to prevent the session manager from displaying the Switch To dialog box. This dialog box gives the user the option to end an application that is holding up the input system. If the debug application does not pass control to the next hook in the chain, it should take action to prevent the faulty application from locking up the input system.

AbnormalInputTimeoutHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

CallHookHook

CallHookHook - Syntax

CallHookHook is called for debugging purposes, whenever any other hook procedure is about to be called.

```
#define INCL_XXX
#include <os2.h>

HMQ      hmqCurrent;    /* Message-queue handle associated with the current thread. */
HMQ      hmqCreated;    /* Message-queue handle created by the installing thread. */
PID      pid;           /* Process identity of the installing thread. */
TID      tid;           /* Thread identity of the installing thread. */
SHORT    sHookType;     /* Hook type. */
PFN      pfnHookProc;   /* Address of the hook procedure that is about to be called. */
BOOL     f;             /* Success indicator. Indicates whether or not the hook pfnHookProc should be called: */

f = CallHookHook(hmqCurrent, hmqCreated, pid,
                 tid, sHookType, pfnHookProc);
```

CallHookHook Parameter - hmqCurrent

hmqCurrent ([HMQ](#)) - input
Message-queue handle associated with the current thread.

CallHookHook Parameter - hmqCreated

hmqCreated ([HMQ](#)) - input
Message-queue handle created by the installing thread.

CallHookHook Parameter - pid

pid ([PID](#)) - input
Process identity of the installing thread.

CallHookHook Parameter - tid

tid ([TID](#)) - input
Thread identity of the installing thread.

CallHookHook Parameter - sHookType

sHookType ([SHORT](#)) - input
Hook type.

This is an HK_* hook-chain.

CallHookHook Parameter - pfnHookProc

pfnHookProc ([PFN](#)) - input
Address of the hook procedure that is about to be called.

CallHookHook Return Value - f

f (BOOL) - returns
Success indicator. Indicates whether or not the hook *pfnHookProc* should be called:

TRUE	The hook referred to in <i>pfnHookProc</i> will not be called.
FALSE	The hook referred to in <i>pfnHookProc</i> will be called.

CallHookHook - Parameters

hmqCurrent ([HMQ](#)) - input
Message-queue handle associated with the current thread.

hmqCreated ([HMQ](#)) - input
Message-queue handle created by the installing thread.

pid ([PID](#)) - input
Process identity of the installing thread.

tid ([TID](#)) - input
Thread identity of the installing thread.

sHookType (SHORT) - input
Hook type.

This is an HK_* hook-chain.

pfnHookProc ([PFN](#)) - input
Address of the hook procedure that is about to be called.

f (BOOL) - returns
Success indicator. Indicates whether or not the hook *pfnHookProc* should be called:

TRUE	The hook referred to in <i>pfnHookProc</i> will not be called.
FALSE	The hook referred to in <i>pfnHookProc</i> will be called.

CallHookHook - Remarks

CallHookHook allows a debug application to catch hook calls to a program that is being debugged.

CallHookHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

Window Functions

This section describes the Window functions that are new or enhanced on OS/2 Warp (PowerPC Edition). The Window functions described in this section are:

- [WinLockInput](#)
- [WinNoShutdown](#)
- [WinQueueFromID](#)
- [WinQuerySendMsg](#)
- [WinReplyMsg](#)
- [WinSetTitleAndHwndIcon](#)
- [WinSetTitleAndIcon](#)
- [WinStretchPointer](#)
- [WinThreadAssocQueue](#)
- [WinWakeThread](#)

WinLockInput

WinLockInput - Syntax

WinLockInput locks or unlocks input to message queues.

```
#define INCL_WIN /* Or use INCL_PM, */
#include <os2.h>

HMQ      hmq;      /* Message-queue handle. */
BOOL     Lock;      /* Lock indicator: */
BOOL     Success;   /* Success indicator. */

Success = WinLockInput(hmq, Lock);
```

WinLockInput Parameter - hmq

hmq ([HMQ](#)) - input
Message-queue handle.

NULL

Lock or unlock all message queues except the one associated with the current thread.

This is the only value accepted for this parameter.

WinLockInput Parameter - Lock

Lock (BOOL) - input
Lock indicator:

TRUE

Lock the message queue or queues, except the current thread's message queue. This is to prevent all threads from generating any more "send" messages to other threads. All mouse and keyboard inputs will be given to the current thread.

FALSE

Unlock the message queue or queues that were locked by a previous call to WinLockInput with *Lock* set to TRUE.

WinLockInput Return Value - Success

Success (BOOL) - returns
Success indicator.

TRUE

Successful completion.

FALSE

An error occurred.

WinLockInput - Parameters

hmq ([HMQ](#)) - input
Message-queue handle.

NULL

Lock or unlock all message queues except the one associated with the current thread.

This is the only value accepted for this parameter.

Lock (BOOL) - input
Lock indicator:

TRUE

Lock the message queue or queues, except the current thread's message queue. This is to prevent all threads from generating any more "send" messages to other threads. All mouse and keyboard inputs will be given to the current thread.

FALSE

Unlock the message queue or queues that were locked by a previous call to WinLockInput with *Lock* set to TRUE.

Success (BOOL) - returns
Success indicator.

TRUE

Successful completion.

FALSE

An error occurred.

WinLockInput - Remarks

A debug application uses this function to lock or unlock input to the message queues of other applications in the system. The debug application uses DosPTrace to give control to, or get control from, the program that is being debugged.

WinLockInput - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinNoShutdown

WinNoShutdown - Syntax

WinNoShutdown prevents a user from closing an OS/2 full-screen or window session that cannot and should not be shut down.

```
#include <os2.h>

HSid      hSid;
FLAG      flag;
BOOL      rc; /* Success indicator. */

rc = WinNoShutdown(hSid, flag);
```

WinNoShutdown Parameter - hSid

hSid (HSid) - input

The session ID of the session that is not to be shut down. If the current session is not to be shut down, specify 0.

WinNoShutdown Parameter - flag

flag (FLAG) - input

Session shutdown indicator, which may be one of the following values:

TRUE	The session will be disabled for shutdown.
FALSE	The session will be enabled for shutdown.

WinNoShutdown Return Value - rc

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion.

FALSE

An error occurred. Also, FALSE is returned if the session ID passed in is associated with a non-OS/2 window.

WinNoShutdown - Parameters

hSid (HSid) - input

The session ID of the session that is not to be shut down. If the current session is not to be shut down, specify 0.

flag (FLAG) - input

Session shutdown indicator, which may be one of the following values:

TRUE

The session will be disabled for shutdown.

FALSE

The session will be enabled for shutdown.

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion.

FALSE

An error occurred. Also, FALSE is returned if the session ID passed in is associated with a non-OS/2 window.

WinNoShutdown - Remarks

When the flag is TRUE:

- For an OS/2 full-screen or window session, the shell will not ask the user if the session is to be shut down and will not issue DosKillProcess for the session. Additionally, the **Close** item on the system menu for the session is disabled.

When the flag is FALSE:

- For an OS/2 full-screen or window session, the shell will re-enable the shutdown. Thus, if **Shut down** is selected, the session can be shut down. The **Close** item on the system menu for the session is enabled.
-

WinNoShutdown - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

WinQueueFromID

WinQueueFromID - Syntax

WinQueueFromID returns the handle of the queue that was created by the specified thread.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB    hab; /* Anchor-block handle. */
PID    pid; /* Process identity. */
TID    tid; /* Thread identity. */
HMQ    hmq; /* Queue handle. */

hmq = WinQueueFromID(hab, pid, tid);
```

WinQueueFromID Parameter - hab

hab (HAB) - input
Anchor-block handle.

WinQueueFromID Parameter - pid

pid (PID) - input
Process identity.

WinQueueFromID Parameter - tid

tid (TID) - input
Thread identity.

WinQueueFromID Return Value - hmq

hmq (HMQ) - returns
Queue handle.

NULL

The thread has not created a message queue.

OTHER

Message-queue handle of the message queue created by the thread specified.

WinQueueFromID - Parameters

hab (HAB) - input
Anchor-block handle.

pid (PID) - input
Process identity.

tid (TID) - input
Thread identity.

hmq (HMQ) - returns
Queue handle.

NULL

The thread has not created a message queue.

OTHER

Message-queue handle of the message queue created by the thread specified.

WinQueueFromID - Remarks

WinQueueFromID does not consider any associations set up by [WinThreadAssocQueue](#).

WinQueueFromID - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinQuerySendMsg

WinQuerySendMsg - Syntax

WinQuerySendMsg returns the first message sent from any window on a specified thread to any window on another specified thread, but has not yet been processed by the receiving window.

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>
```

```

HAB      hab;          /* Anchor-block handle of the caller. */
HMQ      hmqSender;    /* Message-queue handle of the sender. */
HMQ      hmqReceiver;  /* Message-queue handle of the receiver. */
PQMSG    pQmsg;        /* Pointer to the message structure. */
HMQ      hmq;          /* Queue handle of the sender. */

hmq = WinQuerySendMsg(hab, hmqSender, hmqReceiver,
                     pQmsg);

```

WinQuerySendMsg Parameter - hab

hab (HAB) - input
Anchor-block handle of the caller.

WinQuerySendMsg Parameter - hmqSender

hmqSender (HMQ) - input
Message-queue handle of the sender.

NULL
Search for the first message sent from any window on any queue to any window specified by *hmqReceiver* . This value is not valid if *hmqReceiver* also has the value NULL.

OTHER
Search for the first message sent from any window on this queue to any window specified by *hmqReceiver* .

WinQuerySendMsg Parameter - hmqReceiver

hmqReceiver (HMQ) - input
Message-queue handle of the receiver.

NULL
Search for the first message sent to any window on any queue from any window specified by *hmqSender* . This value is not valid if *hmqSender* also has the value NULL.

OTHER
Search for the first message sent to any window on this queue from any window specified by *hmqSender* .

WinQuerySendMsg Parameter - pQmsg

pQmsg (PQMSG) - output
Pointer to the message structure.

Location to store the queried message.

WinQuerySendMsg Return Value - hmq

hmq (HMQ) - returns

Queue handle of the sender.

NULL

Either no message was found that satisfies the search criteria, or an error occurred.

OTHER

Queue handle of the sending window.

WinQuerySendMsg - Parameters

hab (HAB) - input

Anchor-block handle of the caller.

hmqSender (HMQ) - input

Message-queue handle of the sender.

NULL

Search for the first message sent from any window on any queue to any window specified by *hmqReceiver* . This value is not valid if *hmqReceiver* also has the value NULL.

OTHER

Search for the first message sent from any window on this queue to any window specified by *hmqReceiver* .

hmqReceiver (HMQ) - input

Message-queue handle of the receiver.

NULL

Search for the first message sent to any window on any queue from any window specified by *hmqSender* . This value is not valid if *hmqSender* also has the value NULL.

OTHER

Search for the first message sent to any window on this queue from any window specified by *hmqSender* .

pQmsg (PQMSG) - output

Pointer to the message structure.

Location to store the queried message.

hmq (HMQ) - returns

Queue handle of the sender.

NULL

Either no message was found that satisfies the search criteria, or an error occurred.

OTHER

Queue handle of the sending window.

WinQuerySendMsg - Remarks

A debug application uses WinQuerySendMsg to process messages sent to the program it is debugging. The call returns whether or not the thread of the program is currently processing the message. If it is, the thread is waiting for the scheduler to give it another time slice.

Applications that have sent messages to the process of the program being debugged normally have to wait until the program processes those messages. This means that, if the program is currently waiting at a breakpoint, these applications also have to wait. A debug application uses the WinQuerySendMsg function to find out which *sent* messages are currently waiting to be processed. It then uses [WinReplyMsg](#) to release the sending applications.

WinQuerySendMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinReplyMsg

WinReplyMsg - Syntax

WinReplyMsg replies to a message sent from any window on a specified thread to any window on another specified thread, but that has not yet been processed by the receiving window.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HMQ      hmqSender;    /* Message-queue handle of the sender. */
HMQ      hmqReceiver;  /* Message-queue handle of the receiver. */
MRESULT  Reply;        /* Reply passed to the sending window as the return parameter of WinSendMsg. */
BOOL      Success;     /* Success indicator. */

Success = WinReplyMsg(hab, hmqSender, hmqReceiver,
                     Reply);
```

WinReplyMsg Parameter - hab

hab (HAB) - input
Anchor-block handle.

WinReplyMsg Parameter - hmqSender

hmqSender ([HMQ](#)) - input
Message-queue handle of the sender.

NULL	Reply to the first message sent from any window on any queue to any window specified by <i>hmqReceiver</i> . This value is not valid if <i>hmqReceiver</i> also has the value NULL.
OTHER	Reply to the first message sent from any window on this queue to any window specified by <i>hmqReceiver</i> .

WinReplyMsg Parameter - hmqReceiver

hmqReceiver (HMQ) - input

Message-queue handle of the receiver.

NULL

Reply to the first message sent to any window on any queue from any window specified by *hmqSender* . This value is not valid if *hmqSender* also has the value NULL.

OTHER

Reply to the first message sent to any window on this queue from any window specified by *hmqReceiver* .

WinReplyMsg Parameter - Reply

Reply (MRESULT) - input

Reply passed to the sending window as the return parameter of WinSendMessage.

WinReplyMsg Return Value - Success

Success (BOOL) - returns

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

WinReplyMsg - Parameters

hab (HAB) - input

Anchor-block handle.

hmqSender (HMQ) - input

Message-queue handle of the sender.

NULL

Reply to the first message sent from any window on any queue to any window specified by *hmqReceiver* . This value is not valid if *hmqReceiver* also has the value NULL.

OTHER

Reply to the first message sent from any window on this queue to any window specified by *hmqReceiver* .

hmqReceiver (HMQ) - input

Message-queue handle of the receiver.

NULL

Reply to the first message sent to any window on any queue from any window specified by *hmqSender* . This value is not valid if *hmqSender* also has the value NULL.

OTHER

Reply to the first message sent to any window on this queue from any window specified by *hmqReceiver* .

Reply ([MRESULT](#)) - input
Reply passed to the sending window as the return parameter of WinSendMessage.

Success (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinReplyMsg - Remarks

WinReplyMsg is used by a debug application to process messages sent to the program it is debugging.

The call works whether or not the thread of the program being debugged is currently processing the message. If a thread is currently processing a message, it waits for the scheduler to give it another time slice. It resumes processing the sent message when the debug application releases it, but the reply it makes to the message is discarded.

WinReplyMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinSetTitleAndHwndIcon

WinSetTitleAndHwndIcon - Syntax

WinSetTitleAndHwndIcon sets the program title and icon.

```
#include <os2.h>

PSZ      pszTitle;
HWND     hwndIcon;
APIRET   rc;      /* Success indicator. */

rc = WinSetTitleAndHwndIcon(pszTitle, hwndIcon);
```

WinSetTitleAndHwndIcon Parameter - pszTitle

pszTitle (PSZ) - input
Pointer to the program title.

WinSetTitleAndHwndIcon Parameter - hwndIcon

hwndIcon (HWND) - input
Icon handle.

Setting a NULL *hwndIcon* clears out any custom icon set for the task and restores the original default icon.

WinSetTitleAndHwndIcon Return Value - rc

rc (APIRET) - returns
Success indicator.

TRUE	Successful.
FALSE	Not Successful.

WinSetTitleAndHwndIcon - Parameters

pszTitle (PSZ) - input
Pointer to the program title.

hwndIcon (HWND) - input
Icon handle.

Setting a NULL *hwndIcon* clears out any custom icon set for the task and restores the original default icon.

rc (APIRET) - returns
Success indicator.

TRUE	Successful.
FALSE	Not Successful.

WinSetTitleAndHwndIcon - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

WinSetTitleAndIcon

WinSetTitleAndIcon - Syntax

WinSetTitleAndIcon sets the program title and program name. If an icon with the same name as the program name is located, it will be used as well.

```
#include <os2.h>

PSZ      pszTitle;
PSZ      pszProgramName;
APIRET   rc;      /* Success indicator. */

rc = WinSetTitleAndIcon(pszTitle, pszProgramName);
```

WinSetTitleAndIcon Parameter - pszTitle

pszTitle (PSZ) - input
Pointer to the program title.

WinSetTitleAndIcon Parameter - pszProgramName

pszProgramName (PSZ) - input
Pointer to the program name.

WinSetTitleAndIcon Return Value - rc

rc (APIRET) - returns
Success indicator.

TRUE	Successful.
FALSE	Not Successful.

WinSetTitleAndIcon - Parameters

pszTitle (PSZ) - input
Pointer to the program title.

pszProgramName (PSZ) - input
Pointer to the program name.

rc (APIRET) - returns
Success indicator.

TRUE	Successful.
FALSE	Not Successful.

WinSetTitleAndIcon - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

WinStretchPointer

WinStretchPointer - Syntax

WinStretchPointer draws an icon at the *x* and *y* coordinates using the style specified in the *fs* parameter. This function also stretches the bitmap in size using the *cx* and *cy* coordinates.

```
#include <os2.h>

HPS      hps;      /* Presentation-space handle. */
SHORT    x;        /* x coordinate. */
SHORT    y;        /* y coordinate. */
SHORT    cx;       /* cx coordinate. */
SHORT    cy;       /* cy coordinate. */
HPOINTER hIcon;    /* Icon handle. */
USHORT   fs;       /* Style. */
APIRET   rc;       /* Success indicator. */

rc = WinStretchPointer(hps, x, y, cx, cy,
                      hIcon, fs);
```

WinStretchPointer Parameter - hps

hps (HPS) - input
Presentation-space handle.

WinStretchPointer Parameter - x

x (SHORT) - input
x coordinate.

WinStretchPointer Parameter - y

y (SHORT) - input
y coordinate.

WinStretchPointer Parameter - cx

cx (SHORT) - input
cx coordinate.

WinStretchPointer Parameter - cy

cy (SHORT) - input
cy coordinate.

WinStretchPointer Parameter - hIcon

hIcon (HPOINTER) - input
Icon handle.

WinStretchPointer Parameter - fs

fs (USHORT) - input
Style.

The style can be a combination of the following:

DP_NORMAL

Default method.

DP_HALFTONED

Grey - also known as partialized.

DP_INVERTED

Inverted - black is white and white is black.

WinStretchPointer Return Value - rc

rc (APIRET) - returns
Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

WinStretchPointer - Parameters

hps (HPS) - input
Presentation-space handle.

x (SHORT) - input
x coordinate.

y (SHORT) - input
y coordinate.

cx (SHORT) - input
cx coordinate.

cy (SHORT) - input
cy coordinate.

hicon (HPOINTER) - input
Icon handle.

fs (USHORT) - input
Style.

The style can be a combination of the following:

DP_NORMAL

Default method.

DP_HALFTONED

Grey - also known as partialized.

DP_INVERTED

Inverted - black is white and white is black.

rc (APIRET) - returns
Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

WinStretchPointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

WinThreadAssocQueue

WinThreadAssocQueue - Syntax

WinThreadAssocQueue associates the thread of a debug application with, or disassociates it from, the message queue of the program that is being debugged.

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HMQ      hmq;          /* Message-queue handle. */
BOOL      Success;     /* Success indicator. */

Success = WinThreadAssocQueue(hab, hmq);
```

WinThreadAssocQueue Parameter - hab

hab (HAB) - input
Anchor-block handle.

WinThreadAssocQueue Parameter - hmq

hmq (**HMQ**) - input
Message-queue handle.

NULL

Disassociate the current thread from the currently-associated message queue.

The thread is automatically re-associated with the queue it created, if that queue is currently disassociated. The queue is automatically re-associated with the thread that created it, if that thread is currently disassociated. This function does not disassociate a thread from the queue it created.

OTHER

Associate the current thread with the specified queue. Any implicit disassociations obey the rules given above.

WinThreadAssocQueue Return Value - Success

Success (BOOL) - returns
Success indicator.

TRUE

Successful completion.

FALSE

An error occurred.

WinThreadAssocQueue - Parameters

hab (HAB) - input
Anchor-block handle.

hmq (HMQ) - input
Message-queue handle.

NULL

Disassociate the current thread from the currently-associated message queue.

The thread is automatically re-associated with the queue it created, if that queue is currently disassociated. The queue is automatically re-associated with the thread that created it, if that thread is currently disassociated. This function does not disassociate a thread from the queue it created.

OTHER

Associate the current thread with the specified queue. Any implicit disassociations obey the rules given above.

Success (BOOL) - returns
Success indicator.

TRUE

Successful completion.

FALSE

An error occurred.

WinThreadAssocQueue - Remarks

WinThreadAssocQueue is used by a debug application to process messages on behalf of the program it is debugging. The debug application uses DosPTrace to give control to, or get control from, the program.

A thread always retains an association with the queue it created. An association between a thread and a queue it did not create is temporary.

A thread automatically reassociates itself with its own queue if both thread and queue would otherwise be left unassociated as a result of the WinThreadAssocQueue function. For example, consider this sequence of events:

1. A thread creates queue A.
2. The thread associates itself with queue B.
3. The thread associates itself with queue C.
4. The thread disassociates itself using WinThreadAssocQueue with *hmq* set to NULL.

If queue A is currently not associated with any thread, the thread is automatically reassociated with queue A to prevent the thread from becoming queueless.

Messages can still be posted to the queue, but nothing can be removed from the queue until an association is reestablished. This means that threads that have created windows should not use WinThreadAssocQueue; otherwise, the input system might lock up when it tries to pass a keyboard message or mouse message to a queue without a thread.

WinThreadAssocQueue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinWakeThread

WinWakeThread - Syntax

WinWakeThread wakes a thread that is waiting for input, so that a DosPtrace STOP command can be processed.

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HMQ      hmq;          /* Message-queue handle of the thread to be scheduled. */
BOOL      Success;     /* Success indicator. */

Success = WinWakeThread(hmq);
```

WinWakeThread Parameter - hmq

hmq (**HMQ**) - input
Message-queue handle of the thread to be scheduled.

WinWakeThread Return Value - Success

Success (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	An error occurred.

WinWakeThread - Parameters

hmq (**HMQ**) - input
Message-queue handle of the thread to be scheduled.

Success (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	An error occurred.

WinWakeThread - Remarks

A debug application might want to stop a program at a breakpoint, using the DosPtrace STOP command. This command stops a thread only when the thread is scheduled. WinWakeThread forces a thread to be rescheduled, even though it is waiting for input.

WinWakeThread - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

PM Data Types

This section describes the data types that are used with the PM functions, and includes the following data types:

- [MRESULT](#)
- [PID](#)
- [QMSG](#)
- [TID](#)

HMQ

Message-queue handle.

```
typedef LHANDLE HMQ;
```

MRESULT

A 4-byte message-dependent reply parameter structure.

```
typedef VOID *MRESULT;
```

Certain elements of information, placed into the parameters of a message, have data types that do not use all four bytes of this data type. The rules governing these cases are:

BOOL	The value is contained in the low word and the high word is 0.
SHORT	The value is contained in the low word and its sign is extended into the high word.
USHORT	The value is contained in the low word and the high word is 0.
NULL	The entire four bytes are 0.

The structure of this data type depends on the message. For details, see the description of the particular message.

PFN

Pointer to a procedure.

```
typedef _PFN *PFN;
```

In the header file, this is a two-part definition as shown below:

```
typedef int (APIENTRY _PFN) ();  
typedef _PFN *PFN;
```

PID

Process identity.

```
typedef LHANDLE PID;
```

QMSG

Message structure.

```
typedef struct _QMSG {  
    HWND      hwnd;      /* Window handle. */  
    ULONG      msg;       /* Message identity. */  
    MPARAM     mp1;       /* Parameter 1. */  
    MPARAM     mp2;       /* Parameter 2. */  
    ULONG      time;      /* Message time. */  
    POINTL     ptl;       /* Pointer position when message was generated. */  
    ULONG      reserved;  /* Reserved. */  
} QMSG;  
  
typedef QMSG *PQMSG;
```

QMSG Field - hwnd

hwnd (HWND)
Window handle.

QMSG Field - msg

msg (ULONG)
Message identity.

QMSG Field - mp1

mp1 (MPARAM)
Parameter 1.

QMSG Field - mp2

mp2 (MPARAM)
Parameter 2.

QMSG Field - time

time (ULONG)
Message time.

QMSG Field - ptl

ptl (POINTL)
Pointer position when message was generated.

QMSG Field - reserved

reserved (ULONG)
Reserved.

TID

Thread identity.

```
typedef LHANDLE TID;
```

Input Method

This chapter describes new functions and Presentation Manager messages to control the Input Method program.

The Input Method program (IM) is a dynamically loadable and executable module. It gets a KeyEvent packet from Presentation Manager (PM) and converts it to a character set meaningful to applications and users. For instance, in the Japanese version, IM converts Hiragana characters to Kanji characters.

IM generates a character set based on a sequence entered by the user through a candidate window. This character set is then fed back to PM for converting it to WM_CHAR messages. The translated character set is then returned to the application through WM_CHAR messages.

For example, a user enters "alpha" from an application and requests an IM to convert this to Greek representation. The IM generates " Ó " and feeds it back to PM, which in turn sends " Ó " back to the application with a WM_CHAR message.

The functions and messages described in this chapter control the function and behavior of the Input Method program.

The chapter includes the following sections:

- [IM Functions](#)
 - [IM Notification Messages](#)
 - [IM Notification Codes](#)
 - [IM Data Types](#)
-

IM Functions

The Input Method functions described in this section are:

- [WinIMControlMode](#)
 - [WinIMSendCmd](#)
 - [WinIMQueryIM](#)
 - [WinIMSelectIM](#)
-

WinIMControlMode

WinIMControlMode - Syntax

This function queries and sets a Frame keyboard mode.

```
#include <os2.h>

HWND      hwndFrame;
PIMCONTROL pIMControl;
BOOL      rc;

rc = WinIMControlMode(hwndFrame, pIMControl);
```

WinIMControlMode Parameter - hwndFrame

hwndFrame (HWND) - input
Frame window handle to be processed. (NULL indicates a focused window.)

WinIMControlMode Parameter - pIMControl

pIMControl (PIMCONTROL) - input
Pointer to the IMCONTROL structure.

WinIMControlMode Return Value - rc

rc (BOOL) - returns
Return codes.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError:
PMERR_INVALID_PARAMETERS (0x1208) An invalid parameter was specified.
PMERR_INVALID_BUFFER_SIZE (0x110A) An invalid buffer size was specified.
PMERR_INVALID_HWND (0x1101) An invalid window handle was specified.
PMERR_FUNCTION_NOT_SUPPORTED (0x1641) This function is not supported.

WinIMControlMode - Parameters

hwndFrame (HWND) - input
Frame window handle to be processed. (NULL indicates a focused window.)

pIMControl (PIMCONTROL) - input
Pointer to the IMCONTROL structure.

rc (BOOL) - returns
Return codes.

TRUE

Successful completion.

FALSE

Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARAMETERS (0x1208) An invalid parameter was specified.

PMERR_INVALID_BUFFER_SIZE (0x110A) An invalid buffer size was specified.

PMERR_INVALID_HWND (0x1101) An invalid window handle was specified.

PMERR_FUNCTION_NOT_SUPPORTED (0x1641) This function is not supported.

WinIMControlMode - Remarks

This function consolidates the current functions of WinDBCSIMEControl and WinDBCSModeControl functions.

IM_CTL_MOD_ROMANENABLE/DISABLE and IM_CTL_MOD_ROMAN/NOROMAN are only valid for Japanese CodePages.

ulKbdMask is valid at setting time (IM_CTL_REQ_SET) and specifies bits of *ulKbdMode* to be affected. *hwndIM* is filled on return at IM_CTL_REQ_QUERY.

hwndInherit must be set when issuing a set request (*ulReqType* = IM_CTL_REQ_SET) with IM_CTL_MOD_INHERIT_ON. For a query request, if a specified frame is in inherit mode, *hwndInherit* is filled with the Inherit Frame handle.

As the result of IM_CTL_REQ_SET, the Input Method Support module GTR (General Transaction Router) may generate WM_CHAR messages and send them to the specified Frame window. The WM_CHAR messages will contain VKeys representing a new KbdMode.

When IM_CTL_MOD_DBCSINBYTE is set, PM sends two WM_CHARs which contain the first and second bytes of a DBCS ASCII character. Otherwise, PM only sends one WM_CHAR message.

IM_CTL_MOD_INTERIMON is usually OFF. IM_CTL_MOD_FINALONLY is the default for all applications except those using the Korean CodePage. With IM_CTL_MOD_FINALONLY, PM and IM only generate finalized character messages.

The mode INTERIMON is used mostly by Hangeulized applications. When INTERIMON set ON, an IM attached to the frame generates both interim and finalized Hangeul characters, if it has such a capability. PM then returns them to the applications with WM_CHAR messages.

In case of an interim character, the Scancode field of the message contains 0xF0. If it's 0xF1, it represents a finalized Hangeul character.

Note: Under INTERIMON mode, IM will not usually display the interim Hangeul characters. Applications have a chance to hide the conversion window explicitly through an IM_SHOWWINDOW message.

WinIMControlMode - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

WinIMSendCmd

WinIMSendCmd - Syntax

This function sends specified commands to IM.

```
#include <os2.h>

HWND      hwndIM;
HWND      hwnd;
ULONG     ulSubCmd;
PVOID     pEventPkt;
MRESULT   mresReply;

mresReply = WinIMSendCmd(hwndIM, hwnd, ulSubCmd,
                        pEventPkt);
```

WinIMSendCmd Parameter - hwndIM

hwndIM (HWND) - input
IM Window handle to be passed.

WinIMSendCmd Parameter - hwnd

hwnd (HWND) - input
Window handle to be processed. (NULL indicates a focused window.)

WinIMSendCmd Parameter - ulSubCmd

ulSubCmd (ULONG) - input
A sub-command of WM_IM_NOTIFYEVENT.

WinIMSendCmd Parameter - pEventPkt

pEventPkt (PVOID) - in/out
Pointer to an event packet (structure) corresponding to ulSubCmd.

WinIMSendCmd Return Value - mresReply

mresReply (**MRESULT**) - returns
Message-return data.

Possible returns from WinGetLastError:

PMERR_INVALID_PARAMETERS	(0x1208) An invalid parameter was specified.
PMERR_INVALID_HWND	(0x1101) An invalid window handle was specified.
PMERR_FUNCTION_NOT_SUPPORTED	(0x1641) Function not supported.

WinIMSendCmd - Parameters

hwndIM (HWND) - input
IM Window handle to be passed.

hwnd (HWND) - input
Window handle to be processed. (NULL indicates a focused window.)

ulSubCmd (ULONG) - input
A sub-command of WM_IM_NOTIFYEVENT.

pEventPkt (PVOID) - in/out
Pointer to an event packet (structure) corresponding to ulSubCmd.

mresReply ([MRESULT](#)) - returns
Message-return data.

Possible returns from WinGetLastError:

PMERR_INVALID_PARAMETERS	(0x1208) An invalid parameter was specified.
PMERR_INVALID_HWND	(0x1101) An invalid window handle was specified.
PMERR_FUNCTION_NOT_SUPPORTED	(0x1641) Function not supported.

WinIMSendCmd - Remarks

Applications can communicate with an attached IM and control it with this function. For example, an application sends the IM a string as its input characters with an IM_CHAR message for each character. It then sends an IM_CONVERT command to tell the IM to convert the characters just sent. The application then gets the converted string as if it comes from a keyboard.

WinIMSendCmd - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinIMQueryIM

WinIMQueryIM - Syntax

This function returns registered IMs or a specified IM information in the system.

```
#include <os2.h>

PULONG      pRecSize;
PIMREC      pIMRecs;
BOOL        rc;

rc = WinIMQueryIM(pRecSize, pIMRecs);
```

WinIMQueryIM Parameter - pRecSize

pRecSize (PULONG) - in/out
Pointer to total buffer size for IMRECs.

WinIMQueryIM Parameter - pIMRecs

pIMRecs (PIMREC) - in/out
Pointer to an array of IMREC structures.

WinIMQueryIM Return Value - rc

rc (BOOL) - returns
Return codes.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError:
PMERR_INVALID_PARAMETERS (0x1208) An invalid parameter was specified.
PMERR_BUFFER_TOO_SMALL (0x110B) The buffer size is too small.

WinIMQueryIM - Parameters

pRecSize (PULONG) - in/out
Pointer to total buffer size for IMRECs.

pIMRecs ([PIMREC](#)) - in/out
Pointer to an array of IMREC structures.

rc (BOOL) - returns
Return codes.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError:
PMERR_INVALID_PARAMETERS (0x1208) An invalid parameter was specified.
PMERR_BUFFER_TOO_SMALL (0x110B) The buffer size is too small.

WinIMQueryIM - Remarks

When information on all IMs is required, first call WinIMQueryIM with NULL in *pRecSize* . The GTR will return the required total buffer size at the address pointed to by *pRecSize* . Then call WinIMQueryIM with the required buffer size in *pRecSize* .

For example:

```
ULONG    pRecSize = 0;
PIMREC   pIMRecs  = 0;

WinIMQueryIM( &pRecSize, &pIMRecs );
pIMRecs = malloc( pRecSize );
WinIMQueryIM( &pRecSize, &pIMRecs );
```

This function requires *ulSize* and *IMName* when information for a specific IM is queried.

If the *hwndIM* is NULL, it means that the IM is not currently loaded. When the IM is loaded, this field contains the window handle.

WinIMQueryIM - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinIMSelectIM

WinIMSelectIM - Syntax

This function switches the currently attached IM to a specified IM for a specified application.

```
#include <os2.h>

HWND      hwnd;
UniChar   *pIMName;
BOOL      rc;

rc = WinIMSelectIM(hwnd, pIMName);
```

WinIMSelectIM Parameter - hwnd

hwnd (HWND) - input
Window handle to be switched. (NULL indicates a focused window.)

WinIMSelectIM Parameter - pIMName

pIMName (UniChar *) - input
Pointer to an IMName to be attached.

WinIMSelectIM Return Value - rc

rc (BOOL) - returns
Return codes.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError:
PMERR_INVALID_PARAMETERS (0x1208) An invalid parameter was specified.
PMERR_INVALID_HWND (0x1101) An invalid window handle was specified.

WinIMSelectIM - Parameters

hwnd (HWND) - input
Window handle to be switched. (NULL indicates a focused window.)

pIMName (UniChar *) - input
Pointer to an IMName to be attached.

rc (BOOL) - returns
Return codes.

TRUE	Successful completion.
------	------------------------

FALSE

Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARAMETERS (0x1208) An invalid parameter was specified.

PMERR_INVALID_HWND (0x1101) An invalid window handle was specified.

WinIMSelectIM - Remarks

When a specified IM is not found or the specified IM doesn't support the specified Frame Code Page, this function returns FALSE.

If *pIMName* is NULL, the IM currently attached to the specified window is detached.

WinIMSelectIM - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

IM Notification Messages

These messages are initiated by an application window to notify the IM of significant events, and by PM to notify the application window of the IM events.

The IM notification messages described in this section are:

- [WM_IM_NOTIFYEVENT](#)
- [WM_IM_IMEVENT](#)

WM_IM_NOTIFYEVENT

WM_IM_NOTIFYEVENT - Syntax

WM_IM_NOTIFYEVENT messages are sent from PM to IM.

```
param1
    ULONG    notifycode /* Notify code. */
```

```
param2
    ULONG    notifyinfo /* Notify code information. */

returns
    MRESULT rc          /* IM return code. */
```

WM_IM_NOTIFYEVENT Field - notifycode

notifycode (ULONG)
Notify code.

IM uses the following notification codes. For the complete description of the specified *notifycode* , see [IM Notification Codes](#).

- IM_CHAR
Send a character.
- IM_SHOWWINDOW
Make IM's conversion window visible or invisible.
- IM_CONVERT
Request IM to perform a conversion.
- IM_EXITIMMODE
Force IM to exit from IM mode.
- IM_WORDREGISTER
Start a WordRegistration session.
- IM_CODEINPUT
Invoke a CodeInput session.
- IM_UIFDIALOG
Start a User I/F dialog.
- IM_FIXCANDIDATE
Finalize the current candidate.
- IM_FLUSH
Flush all intermediate status, including a content of conversion window.
- IM_CONVPOSITION
Inform IM of the current conversion position where IM displays its conversion window.
- IM_CONVFONT
Inform IM of font information.

WM_IM_NOTIFYEVENT Field - notifyinfo

notifyinfo (ULONG)
Notify code information.

For the definition of this parameter, see the description of the specified *notifycode* in [IM Notification Codes](#).

WM_IM_NOTIFYEVENT Return Value - rc

rc ([MRESULT](#))
IM return code.

WM_IM_NOTIFYEVENT - Parameters

notifycode (ULONG)
Notify code.

IM uses the following notification codes. For the complete description of the specified *notifycode* , see [IM Notification Codes](#).

IM_CHAR
Send a character.

IM_SHOWWINDOW
Make IM's conversion window visible or invisible.

IM_CONVERT
Request IM to perform a conversion.

IM_EXITIMMODE
Force IM to exit from IM mode.

IM_WORDREGISTER
Start a WordRegistration session.

IM_CODEINPUT
Invoke a CodeInput session.

IM_UIFDIALOG
Start a User I/F dialog.

IM_FIXCANDIDATE
Finalize the current candidate.

IM_FLUSH
Flush all intermediate status, including a content of conversion window.

IM_CONVPOSITION
Inform IM of the current conversion position where IM displays its conversion window.

IM_CONVFONT
Inform IM of font information.

notifyinfo (ULONG)
Notify code information.

For the definition of this parameter, see the description of the specified *notifycode* in [IM Notification Codes](#).

rc ([MRESULT](#))
IM return code.

WM_IM_NOTIFYEVENT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

WM_IM_IMEVENT

WM_IM_IMEVENT - Syntax

WM_IM_IMEVENT messages are sent from PM to an application.

```
param1
    ULONG    notifycode /* Notify code. */

param2
    ULONG    notifyinfo /* Notify code information. */

returns
    BOOL     rc          /* Application return code. */
```

WM_IM_IMEVENT Field - notifycode

notifycode (ULONG)
Notify code.

The GTR uses the following notification codes. For the complete description of the specified *notifycode* , see [IM Notification Codes](#).

IME_STRPACKET
GTR sends IM's output string with phrase information to a focused window.

IME_STRING
GTR sends IM's output string to a focused window.

IME_STRFIRST
GTR sends this message to a focused window to notify that it is about to send IM's output string with WM_CHAR messages.

IME_STRLAST
GTR sends this message to a focused window to notify that is finished sending IM's output string.

WM_IM_IMEVENT Field - notifyinfo

notifyinfo (ULONG)
Notify code information.

For the definition of this parameter, see the description of the specified *notifycode* in [IM Notification Codes](#).

WM_IM_IMEVENT Return Value - rc

rc (BOOL)
Application return code.

WM_IM_IMEVENT - Parameters

notifycode (ULONG)
Notify code.

The GTR uses the following notification codes. For the complete description of the specified *notifycode* , see [IM Notification Codes](#).

IME_STRPACKET
GTR sends IM's output string with phrase information to a focused window.

IME_STRING
GTR sends IM's output string to a focused window.

IME_STRFIRST
GTR sends this message to a focused window to notify that it is about to send IM's output string with [WM_CHAR](#) messages.

IME_STRLAST
GTR sends this message to a focused window to notify that is finished sending IM's output string.

notifyinfo (ULONG)
Notify code information.

For the definition of this parameter, see the description of the specified *notifycode* in [IM Notification Codes](#).

rc (BOOL)
Application return code.

WM_IM_IMEVENT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

IM Notification Codes

This section describes the following [WM_IM_NOTIFYEVENT](#) notification codes that are sent by the application window to the IM.

- [IM_CHAR](#)
- [IM_CODEINPUT](#)
- [IM_CONVERT](#)
- [IM_CONVFONT](#)
- [IM_CONVPOSITION](#)
- [IM_EXITMODE](#)
- [IM_FIXCANDIDATE](#)
- [IM_FLUSH](#)
- [IM_SHOWWINDOW](#)
- [IM_UIFDIALOG](#)

- [IM_WORDREGISTER](#)
- [IME_STRPACKET](#)
- [IME_STRING](#)
- [IME_STRFIRST](#)
- [IME_STRLAST](#)

IM_CHAR

IM_CHAR - Syntax

Applications send a key packet to IM through this message as if it comes from a keyboard.

```
param1
    ULONG      IM\_CHAR    /* Notification code. */

param2
    PCHARINFO  pCharInfo /* Pointer to the CHARINFO structure. */

returns
    IMRET      RetIM      /* IM return code. */
```

IM_CHAR Field - IM_CHAR

IM_CHAR (ULONG)
Notification code.

IM_CHAR Field - pCharInfo

pCharInfo ([PCHARINFO](#))
Pointer to the [CHARINFO](#) structure.

IM_CHAR Return Value - RetIM

RetIM (IMRET)
IM return code.

0	IM_RC_ACCEPTED
1	IM_CHAR should be removed
	IM_RC_IGNORED
	IM_CHAR should be sent to the application
-1	Error occurred

IM_CHAR - Parameters

IM_CHAR (ULONG)
Notification code.

pCharInfo ([PCHARINFO](#))
Pointer to the [CHARINFO](#) structure.

RetIM (IMRET)
IM return code.

0	IM_RC_ACCEPTED
1	IM_CHAR should be removed
	IM_RC_IGNORED
	IM_CHAR should be sent to the application
-1	Error occurred

IM_CHAR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

IM_CODEINPUT

IM_CODEINPUT - Syntax

Invoke a CodeInput session.

```
param1
    ULONG          IM_CODEINPUT /* Notification code. */

param2
    PFRAMEINFO pFrameInfo /* Pointer to the FRAMEINFO structure. */

returns
    BOOL rc /* Success indicator. */
```

IM_CODEINPUT Field - IM_CODEINPUT

IM_CODEINPUT (ULONG)
Notification code.

IM_CODEINPUT Field - pFrameInfo

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

IM_CODEINPUT Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_CODEINPUT - Parameters

IM_CODEINPUT (ULONG)
Notification code.

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_CODEINPUT - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)

IM_CONVERT

IM_CONVERT - Syntax

A request for IM to perform a conversion.

```
param1
    ULONG      IM_CONVERT /* Notification code. */

param2
    PIMREQCONV pIMReqConv /* Pointer to the IMREQCONV structure. */

returns
    BOOL      rc          /* Success indicator. */
```

IM_CONVERT Field - IM_CONVERT

IM_CONVERT (ULONG)
Notification code.

IM_CONVERT Field - pIMReqConv

pIMReqConv (PIMREQCONV)
Pointer to the IMREQCONV structure.

IM_CONVERT Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_CONVERT - Parameters

IM_CONVERT (ULONG)
Notification code.

pIMReqConv ([PIMREQCONV](#))
Pointer to the [IMREQCONV](#) structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_CONVERT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

IM_CONVFONT

IM_CONVFONT - Syntax

Inform the IM of the font information used by the conversion window.

```
param1
    ULONG      IM_CONVFONT /* Notification code. */

param2
    PCONVFONT  pConvFont   /* Pointer to the CONVFONT structure. */

returns
    BOOL      rc           /* Success indicator. */
```

IM_CONVFONT Field - IM_CONVFONT

IM_CONVFONT (ULONG)

Notification code.

IM_CONVFONT Field - pConvFont

pConvFont ([PCONVFONT](#))
Pointer to the [CONVFONT](#) structure.

IM_CONVFONT Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_CONVFONT - Parameters

IM_CONVFONT (ULONG)
Notification code.

pConvFont ([PCONVFONT](#))
Pointer to the [CONVFONT](#) structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_CONVFONT - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

IM_CONVPOSITION

IM_CONVPOSITION - Syntax

Inform the IM of the current appropriate conversion position where the IM displays its conversion window.

```
param1
    ULONG      IM_CONVPOSITION /* Notification code. */

param2
    PCONVPOS   pConvPos        /* Pointer to the CONVPOS structure. */

returns
    BOOL       rc              /* Success indicator. */
```

IM_CONVPOSITION Field - IM_CONVPOSITION

IM_CONVPOSITION (ULONG)
Notification code.

IM_CONVPOSITION Field - pConvPos

pConvPos (PCONVPOS)
Pointer to the CONVPOS structure.

IM_CONVPOSITION Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_CONVPOSITION - Parameters

IM_CONVPOSITION (ULONG)
Notification code.

pConvPos (PCONVPOS)
Pointer to the CONVPOS structure.

rc (BOOL)
Success indicator.

TRUE
FALSE

Successful completion
Error occurred

IM_CONVPOSITION - Remarks

ConvWndPos is the conversion window position which is mapped to the Focus Window or the Frame Window coordinate upon *usDispMode* .

When IM_CNV_FOCUS_COORD is set in *usDispMode* , *ConvWndPos* and *ConvWndBegin* are mapped to the Focus Window coordinate. If IM_CNV_WND_NULL is specified, applicaitons put NULL in *ConvWndPos* . When IM_CNV_WND_FOCUS or IM_CNV_WND_FRAME is set, an application fills the Focus and the Frame Window coordinates respectively. When IM_CNV_WND_XFCS_YFRM is set, an application fills the Focus Window - X coordinate and the Frame Window - Y coordinate.

ConvWndBegin specifies a starting point in the conversion window from where the IM displays the intermediate string. In most cases, it is same as the cursor position obtained in *pCursorPos* with the WM_QUERYCONVERTPOS message.

Note: The application must set IM_CNV_MSG_FROMAPPL in *usDispMode* . Once the applications send this message, the GTR stops to send IM_CONVPOSITION periodically for the Frame window.

IM_CONVPOSITION - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

IM_EXITMODE

IM_EXITMODE - Syntax

Force IM to exit from its IM mode.

```
param1
    ULONG          IM_EXITMODE /* Notification code. */

param2
    PFRAMEINFO     pFrameInfo /* Pointer to the FRAMEINFO structure. */

returns
    BOOL          rc          /* Success indicator. */
```

IM_EXITMODE Field - IM_EXITMODE

IM_EXITMODE (ULONG)
Notification code.

IM_EXITMODE Field - pFrameInfo

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

IM_EXITMODE Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred - unable to exit now

IM_EXITMODE - Parameters

IM_EXITMODE (ULONG)
Notification code.

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred - unable to exit now

IM_EXITMODE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

IM_FIXCANDIDATE

IM_FIXCANDIDATE - Syntax

Finalize the current candidate.

```
param1
    ULONG          IM_FIXCANDIDATE /* Notification code. */

param2
    PFRAMEINFO     pFrameInfo      /* Pointer to the FRAMEINFO structure. */

returns
    BOOL           rc               /* Success indicator. */
```

IM_FIXCANDIDATE Field - IM_FIXCANDIDATE

IM_FIXCANDIDATE (ULONG)
Notification code.

IM_FIXCANDIDATE Field - pFrameInfo

pFrameInfo (PFRAMEINFO)
Pointer to the FRAMEINFO structure.

IM_FIXCANDIDATE Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_FIXCANDIDATE - Parameters

IM_FIXCANDIDATE (ULONG)

Notification code.

pFrameInfo ([PFRAMEINFO](#))

Pointer to the [FRAMEINFO](#) structure.

rc (BOOL)

Success indicator.

TRUE

FALSE

Successful completion

Error occurred

IM_FIXCANDIDATE - Remarks

If there is no intermediate string to finalize, or the IM is unable to fix a string at this time, the IM returns FALSE.

IM_FIXCANDIDATE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

IM_FLUSH

IM_FLUSH - Syntax

Flush all intermediate status, including a content of conversion window.

```
param1
    ULONG          IM\_FLUSH    /* Notification code. */

param2
    PFRAMEINFO pFrameInfo /* Pointer to the FRAMEINFO structure. */

returns
    BOOL          rc          /* Success indicator. */
```

IM_FLUSH Field - IM_FLUSH

IM_FLUSH (ULONG)
Notification code.

IM_FLUSH Field - pFrameInfo

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

IM_FLUSH Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_FLUSH - Parameters

IM_FLUSH (ULONG)
Notification code.

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_FLUSH - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

IM_SHOWWINDOW

IM_SHOWWINDOW - Syntax

Set IM's conversion window visible or invisible.

```
param1
    ULONG          IM_SHOWWINDOW /* Notification code. */

param2
    PIMCTLWND      pIMCtlWnd      /* Pointer to the IMCTLWND structure. */

returns
    BOOL           rc              /* Success indicator. */
```

IM_SHOWWINDOW Field - IM_SHOWWINDOW

IM_SHOWWINDOW (ULONG)
Notification code.

IM_SHOWWINDOW Field - pIMCtlWnd

pIMCtlWnd (PIMCTLWND)
Pointer to the IMCTLWND structure.

IM_SHOWWINDOW Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_SHOWWINDOW - Parameters

IM_SHOWWINDOW (ULONG)

Notification code.

pIMCtlWnd ([PIMCTLWND](#))

Pointer to the [IMCTLWND](#) structure.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred

IM_SHOWWINDOW - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

IM_UIFDIALOG

IM_UIFDIALOG - Syntax

Start a User I/F dialog.

```
param1
    ULONG          IM\_UIFDIALOG /* Notification code. */

param2
    PFRAMEINFO pFrameInfo /* Pointer to the FRAMEINFO structure. */

returns
    BOOL          rc /* Success indicator. */
```

IM_UIFDIALOG Field - IM_UIFDIALOG

IM_UIFDIALOG (ULONG)

Notification code.

IM_UIFDIALOG Field - pFrameInfo

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

IM_UIFDIALOG Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_UIFDIALOG - Parameters

IM_UIFDIALOG (ULONG)
Notification code.

pFrameInfo ([PFRAMEINFO](#))
Pointer to the [FRAMEINFO](#) structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_UIFDIALOG - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

IM_WORDREGISTER

IM_WORDREGISTER - Syntax

Request IM to start a WordRegistration session.

```

param1
    ULONG      IM_WORDREGISTER /* Notification code. */

param2
    PIMWRDREG  pIMWrdReg      /* Pointer to the IMWRDREG structure. */

returns
    BOOL       rc              /* Success indicator. */

```

IM_WORDREGISTER Field - IM_WORDREGISTER

IM_WORDREGISTER (ULONG)
Notification code.

IM_WORDREGISTER Field - pIMWrdReg

pIMWrdReg (PIMWRDREG)
Pointer to the IMWRDREG structure.

IM_WORDREGISTER Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_WORDREGISTER - Parameters

IM_WORDREGISTER (ULONG)
Notification code.

pIMWrdReg (PIMWRDREG)
Pointer to the IMWRDREG structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred

IM_WORDREGISTER - Remarks

To initiate a WordRegistration session without pre-selected words, put NULL in *pIMWrdReg*

IM_WORDREGISTER - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

IME_STRPACKET

IME_STRPACKET - Syntax

GTR sends IM's output string with phrase information to a focused window.

```
param1
    ULONG      IME_STRPACKET /* Notification code. */

param2
    PSTRPKT    pStrPacket    /* Pointer to the STRPKT structure. */

returns
    BOOL       rc            /* Processed indicator. */
```

IME_STRPACKET Field - IME_STRPACKET

IME_STRPACKET (ULONG)
Notification code.

IME_STRPACKET Field - pStrPacket

pStrPacket (PSTRPKT)
Pointer to the **STRPKT** structure.

IME_STRPACKET Return Value - rc

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored

IME_STRPACKET - Parameters

IME_STRPACKET (ULONG)
Notification code.

pStrPacket (PSTRPKT)
Pointer to the **STRPKT** structure.

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored

IME_STRPACKET - Remarks

The format of part-of-Speech information varies among IMs, so it might only be useful for applications which are tightly coupled with the specified IM, e.g. WordProcessor.

In this release, the character attribute is only used for supporting Hangeul Interim character. The following attribute is defined to distinguish between interim and finalized characters.

Bit 0	1 IM_CHR_ATR_INTERIM	Interim character
	0 IM_CHR_ATR_FINAL	Finalized character
Bits 1-7	Reserved, set to zero	

If there is no string to be returned, a corresponding offset field must contain zero. For example, if the IM doesn't return `InpPhrs[]`, it puts zero in the `usInpPhrsOff` field.

Note: The strings are terminated with zero, i.e. 0x00 in IM_CHR_MOD_ASCII and 0x0000 in IM_CHR_MOD_UNICODE.

If the focused application doesn't accept this message, the GTR sends only the output string to it using an IME_STRING message.

IME_STRPACKET - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

IME_STRING

IME_STRING - Syntax

GTR sends IM's output string to a focused window.

```
param1
    ULONG      IME_STRING  /* Notification code. */

param2
    PIMSTRING  pIMString   /* Pointer to the IMSTRING structure. */

returns
    BOOL       rc          /* Processed indicator. */
```

IME_STRING Field - IME_STRING

IME_STRING (ULONG)
Notification code.

IME_STRING Field - pIMString

pIMString (PIMSTRING)
Pointer to the IMSTRING structure.

IME_STRING Return Value - rc

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored

IME_STRING - Parameters

- IME_STRING** (ULONG)
Notification code.
- pIMString** (PIMSTRING)
Pointer to the IMSTRING structure.
- rc** (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored

IME_STRING - Remarks

In this release, OutAtr[] is only used for supporting Hangeul Interim characters. The following attribute is defined to distinguish between interim and finalized characters:

Bit 0	1 IM_CHR_ATR_INTERIM	Interim character
	0 IM_CHR_ATR_FINAL	Finalized character
Bits 1-7	Reserved, set to zero	

Note: The strings are terminated with zero, i.e. 0x00 in IM_CHR_MOD_ASCII and 0x0000 in IM_CHR_MOD_UNICODE.

If the Focused application doesn't accept this message, the GTR sends the IM's output string using WM_CHAR messages.

IME_STRING - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)

IME_STRFIRST

IME_STRFIRST - Syntax

The GTR sends this message to the focused window prior to sending the IM's output string with WM_CHAR messages.

```
param1
    ULONG    IME_STRFIRST /* Notification code. */

param2
    HWND     hwnd        /* Window handle. */

returns
    BOOL     rc          /* Processed indicator. */
```

IME_STRFIRST Field - IME_STRFIRST

IME_STRFIRST (ULONG)
Notification code.

IME_STRFIRST Field - hwnd

hwnd (HWND)
Window handle.

Reserved, set to NULL.

IME_STRFIRST Return Value - rc

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored

IME_STRFIRST - Parameters

IME_STRFIRST (ULONG)
Notification code.

hwnd (HWND)

Window handle.

Reserved, set to NULL.

rc (BOOL)

Processed indicator.

TRUE

FALSE

Message processed

Message ignored

IME_STRFIRST - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

IME_STRLAST

IME_STRLAST - Syntax

The GTR sends this message to the focused window when it is finished sending the IM's output string with WM_CHAR messages.

```
param1
    ULONG    IME_STRLAST    /* Notification code. */

param2
    HWND     hwnd           /* Window handle. */

returns
    BOOL     rc             /* Processed indicator. */
```

IME_STRLAST Field - IME_STRLAST

IME_STRLAST (ULONG)

Notification code.

IME_STRLAST Field - hwnd

hwnd (HWND)
Window handle.

Reserved, set to NULL.

IME_STRLAST Return Value - rc

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored

IME_STRLAST - Parameters

IME_STRLAST (ULONG)
Notification code.

hwnd (HWND)
Window handle.

Reserved, set to NULL.

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored

IME_STRLAST - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Glossary](#)

IM Data Types

This section describes the following data types that are used with the IM functions.

- [CHARINFO](#)
- [CNVSTR](#)
- [CONVFONT](#)
- [CONVPOS](#)

- FRAMEINFO
- IMCONTROL
- IMCTLWND
- IMREC
- IMREQCONV
- IMSTRING
- IMWRDREG
- STRPKT

CHARINFO

Character information record.

```
typedef struct _CHARINFO {
    ULONG      ulSize;
    HWND       hwnd;
    HWND       hwndFrame;
    KEYEVENT   KeyEvent;
} CHARINFO;

typedef CHARINFO *PCHARINFO;
```

CHARINFO Field - ulSize

ulSize (ULONG)
Size, in bytes, of this data structure.

CHARINFO Field - hwnd

hwnd (HWND)
Window handle to be passed.

CHARINFO Field - hwndFrame

hwndFrame (HWND)
Frame window handle to be passed.

CHARINFO Field - KeyEvent

KeyEvent (KEYEVENT)

CNVSTR

Conversion string structure.

```
typedef struct _CNVSTR {
    ULONG      ulSize;
    ULONG      ulFlag;
    USHORT     usInpStrOff;
    USHORT     usInpPhrsOff;
    USHORT     usOutStrOff;
    USHORT     usOutAtrOff;
    USHORT     usOutPhrsOff;
    USHORT     usOutPosOff;
    USHORT     InOutStr[1];
} CNVSTR;

typedef CNVSTR *PCNVSTR;
```

CNVSTR Field - ulSize

ulSize (ULONG)
Size, in bytes, of this data structure.

CNVSTR Field - ulFlag

ulFlag (ULONG)
Indicates the character coding scheme of the input and output string buffers, as follows:

Bit 0	1	IM_CHR_MOD_ASCII
	0	IM_CHR_MOD_UNICODE
Bits 1-31	Reserved, set to zero	

CNVSTR Field - usInpStrOff

usInpStrOff (USHORT)
Offset to the input string buffer.

CNVSTR Field - usInpPhrsOff

usInpPhrsOff (USHORT)
Offset to the input string phrase information buffer.

CNVSTR Field - usOutStrOff

usOutStrOff (USHORT)
Offset to the output string buffer.

CNVSTR Field - usOutAtrOff

usOutAtrOff (USHORT)
Offset to the output string attribute buffer.

CNVSTR Field - usOutPhrsOff

usOutPhrsOff (USHORT)
Offset to the output string phrase information buffer.

CNVSTR Field - usOutPosOff

usOutPosOff (USHORT)
Offset to the output string part-of-speech information buffer.

CNVSTR Field - InOutStr[1]

InOutStr[1] (USHORT)
Consists of the following buffers:

BYTE	InpStr[]	Input string
BYTE	InpPhrs[]	Phrase information
BYTE	OutStr[]	Output string
BYTE	OutAtr[]	Character attribute
BYTE	OutPhrs[]	Phrase information
BYTE	OutPos[]	Part-of-speech information

CONVFONT

Font information used by the IM conversion window.

```
typedef struct _CONVFONT {
    ULONG      ulsize;          /* Size, in bytes, of this data structure. */
    HWND       hwndFocus;      /* Focus window handle to be processed. */
    HWND       hwndFrame;
    PSZ        pszFontData;
} CONVFONT;

typedef CONVFONT *PCONVFONT;
```

CONVFONT Field - ulsize

ulsize (ULONG)
Size, in bytes, of this data structure.

CONVFONT Field - hwndFocus

hwndFocus (HWND)
Focus window handle to be processed.

CONVFONT Field - hwndFrame

hwndFrame (HWND)
Frame window handle associated with the *hwndFocus* .

CONVFONT Field - pszFontData

pszFontData (PSZ)
Same format as PP_FONTNAMESIZE (for example, "12.Helv").
See PP_FONTNAMESIZE in *Presentation Manager Programming Reference* Vol I.

CONVPOS

Conversion position data structure.

```
typedef struct _CONVPOS {
    ULONG      ulsize;          /* Size, in bytes, of this data structure. */
    HWND       hwndFocus;      /* Focus window handle to be processed. */
    HWND       hwndFrame;
    USHORT     usCode;         /* Conversion code. */
    USHORT     usDispMode;
    RECTL      ConvWndPos;     /* Position of the IM's conversion window. */
    RECTL      ConvWndBegin;   /* Starting position to display a conversion string. */
} CONVPOS;

typedef CONVPOS *PCONVPOS;
```

CONVPOS Field - ulsize

ulsize (ULONG)
Size, in bytes, of this data structure.

CONVPOS Field - hwndFocus

hwndFocus (HWND)
Focus window handle to be processed.

CONVPOS Field - hwndFrame

hwndFrame (HWND)
Frame window handle associated with the *hwndFocus* .

CONVPOS Field - usCode

usCode (USHORT)
Conversion code.

Retained for compatability with OS/2 legacy applications. See WM_QUERYCONVERTPOS in *Presentation Manager Programming Reference* Vol II.

CONVPOS Field - usDispMode

usDispMode (USHORT)
Specifies how the IM should display intermediate strings within a specified window position.

The following display modes are defined:

Bit 0	Specify vertical or horizontal representation
1	IM_CNV_DSP_VERTICAL
0	IM_CNV_DSP_HORIZONTAL
Bit 1	Specify direction of characters .
1	IM_CNV_DSP_RIGHT2LEFT
0	IM_CNV_DSP_LEFT2RIGHT
Bit 2-7	Reserved, set to zero
Bit 8	Specify screen or focus window coordinate
1	IM_CNV_FOCUS_COORD
0	IM_CNV_SCRN_COORD
Bit 9-10	Specify a format of <i>ConvWndPos</i>
00	IM_CNV_WND_NULL
01	IM_CNV_WND_FOCUS
10	IDM_CNV_WND_FRAME
11	IDM_CNV_WND_XFCS_YFRM
Bit 11-14	Reserved, set to zero
Bit 15	Specify a sender
1	IDM_CNV_MSG_FROMAPPL
0	IDM_CNV_MSG_FROMGTR

CONVPOS Field - ConvWndPos

ConvWndPos (RECTL)
Position of the IM's conversion window.

CONVPOS Field - ConvWndBegin

ConvWndBegin (RECTL)
Starting position to display a conversion string.

FRAMEINFO

FRAMEINFO data structure.

```
typedef struct _FRAMEINFO {
    ULONG      ulSize;
    HWND       hwndFrame;
} FRAMEINFO;

typedef FRAMEINFO *PFRAMEINFO;
```

FRAMEINFO Field - ulSize

ulSize (ULONG)
Size, in bytes, of this data structure.

FRAMEINFO Field - hwndFrame

hwndFrame (HWND)
Frame window handle to be processed.

IMCONTROL

Input Method Control data structure.

```
typedef struct _IMCONTROL {
    ULONG      ulsize;      /* Size, in bytes, of this data structure. */
    ULONG      ulReqType;   /* Request type. */
    HWND       hwndIM;      /* IM window handle associated to the FrameWindow. */
    HWND       hwndInherit; /* FrameWindow handle to be inherited. */
    ULONG      ulKbdMode;   /* Keyboard mode of the Frame. */
    ULONG      ulKbdMask;   /* Mask to specify effective KbdMode bits. */
} IMCONTROL;

typedef IMCONTROL *PIMCONTROL;
```

IMCONTROL Field - ulsize

ulsize (ULONG)
Size, in bytes, of this data structure.

IMCONTROL Field - ulReqType

ulReqType (ULONG)
Request type.

May be one of the following values:

IM_CTL_REQ_QUERY
IM_CTL_REQ_SET

On return, ulKbdMode is filled.
Value in ulKbdMode is set by the caller.

IMCONTROL Field - hwndIM

hwndIM (HWND)
IM window handle associated to the FrameWindow.

IMCONTROL Field - hwndInherit

hwndInherit (HWND)
FrameWindow handle to be inherited.

IMCONTROL Field - ulKbdMode

ulKbdMode (ULONG)
Keyboard mode of the Frame.

IMMode - IM related modes

Bit 0	Has the following values:
1	IM_CTL_MOD_IMENABLE Make IM enabled
0	IM_CTL_MOD_IMDISABLE Make IM disabled
Bit 1	Has the following values:
1	IM_CTL_MOD_IMMODE Get into IM mode
0	IM_CTL_MOD_NOIMMODE Get out of IM mode
Bit 2	Has the following values:
1	IM_CTL_MOD_ROMANENABLE Enable Roman support
0	IM_CTL_MOD_ROMANDISABLE Disable Roman support
Bit 3-7	Reserved, set to 0
IMLayer - related to IM Kbd layers	
Bit 8	Has the following values:
1	IM_CTL_MOD_WIDECHAR Return wide character
0	IM_CTL_MOD_NARROWCHAR Return normal character
Bits 9-10	Has the following values:
00	IM_CTL_MOD_NLS1 NLS Layer 1
01	IM_CTL_MOD_NLS2

	10	NLS Layer 2 IM_CTL_MOD_NLS3 NLS Layer 3
	11	IM_CTL_MOD_NLS4 NLS Layer 4
Bit 11	Has the following values:	
	1	IM_CTL_MOD_ROMAN Get into Roman mode
	0	IM_CTL_MOD_NOROMAN Get out from Roman mode
Bit 12-15	Reserved, set to 0	
PktMode - related to IM Packet mode		
Bit 16	Has the following values:	
	1	IM_CTL_MOD_DBCSINBYTE Two WM_CHARS represent a DBCS ASCII character.
	0	IM_CTL_MOD_DBCSINWORD One WM_CHAR contains a DBCS ASCII character.
Bit 17	Has the following values:	
	1	IM_CTL_MOD_INTERIMON PMWIN (or IM) returns both interim and final character packets
	0	IM_CTL_MOD_FINALONLY PMWIN (or IM) returns only finalized (converted) packets
Bits 18-23	Reserved, set to 0	
Bits 24-27	Reserved for system use	
Bits 24-27	Reserved, set to 0	
Bit 31	Has the following values:	
	1	IM_CTL_MOD_INHERIT_ON Inherit other Frame
	0	IM_CTL_MOD_INHERIT_OFF Don't inherit

IMCONTROL Field - ulKbdMask

ulKbdMask (ULONG)

Mask to specify effective KbdMode bits.

Bit 0	IM_CTL_MASK_IMENABLE
Bit 1	IM_CTL_MASK_IMMODE
Bit 2	IM_CTL_MASK_ROMANENABLE
Bit 8	IM_CTL_MASK_CHARSIZE
Bits 9-10	IM_CTL_MASK_NLSLAYERS
Bit 11	IM_CTL_MASK_ROMANMODE
Bit 16	IM_CTL_MASK_DBCSWMCHAR
Bit 17	IM_CTL_MASK_INTERIMFINAL
Bit 31	IM_CTL_MASK_INHERITMODE

QueueCP= Japanese CodePages

NLS1	IM_CTL_MOD_ALPHANUMERIC
NLS2	IM_CTL_MOD_KATAKANA
NLS3	IM_CTL_MOD_HIRAGANA

QueueCP= Korean CodePages

NLS1	IM_CTL_MOD_ALPHANUMERIC
NLS2	IM_CTL_MOD_JAMO
NLS3	IM_CTL_MOD_HANGEUL

QueueCP= Taiwanese CodePages

NLS1	IM_CTL_MOD_ALPHANUMERIC
NLS2	IM_CTL_MOD_TSANGJYE
NLS3	IM_CTL_MOD_PHONETIC

QueueCP= Chinese CodePages NLS1, NLS2 and NLS3 are to be defined.

IMCTLWND

IMCTLWND data structure.

```
typedef struct _IMCTLWND {
    ULONG      ulSize;
    HWND       hwndFrame;
    ULONG      ulCtlWnd;
} IMCTLWND;

typedef IMCTLWND *PIMCTLWND;
```

IMCTLWND Field - ulSize

ulSize (ULONG)
Size, in bytes, of this data structure.

IMCTLWND Field - hwndFrame

hwndFrame (HWND)
Frame window handle to be affected.

IMCTLWND Field - ulCtlWnd

ulCtlWnd (ULONG)
Action to be performed. May be one of the following actions:

IM_CTL_WND_SHOWALL	Show all conversion and status windows
--------------------	--

IM_CTL_WND_HIDEALL	Hide all conversion and status windows
IM_CTL_WND_SHOWINDICATOR	Show status indicator
IM_CTL_WND_HIDEINDICATOR	Hide status indicator
IM_CTL_WND_SHOWCONVWNDS	Show all conversion windows
IM_CTL_WND_HIDECONVWNDS	Hide all conversion windows
IM_CTL_WND_TGLINDICATOR	Flash indicator mode

IMREC

Input Method (IM) record.

```
typedef struct _IMREC {
    ULONG      ulSize;
    ULONG      ulFlag;
    FDATE      InstDate;
    FTIME      InstTime;
    USHORT     usCodePages[16];
    UniChar    LangId[2];
    ULONG      ulIMVersion;
    HWND       hwndIM;
    UniChar    IMName[16];
    UniChar    IMModName[64];
    UniChar    VIMName[64];
    UniChar    VIMStubName[64];
    UniChar    IMDescription[64];
    UniChar    IMOptions[64];
} IMREC;

typedef IMREC *PIMREC;
```

IMREC Field - ulSize

ulSize (ULONG)
Size, in bytes, of this data structure.

IMREC Field - ulFlag

ulFlag (ULONG)
Flag that indicates the IM's characteristics and status.

IM_REC_SYSTEM_IM	0x00000001
IM_REC_DEFAULT_IM	0x00000002

IM_REC_STUB_TYPE_IAS_COMPLIANT
0x00000100
IM_REC_STUB_TYPE_DOS_DD
0x00000200
IM_REC_LOADINGIM
0x01000000
IM_REC_LOADEDIM
0x02000000
IM_REC_ACTIVEIM
0x04000000

IMREC Field - InstDate

InstDate (FDATE)
IM installation date.

IMREC Field - InstTime

InstTime (FTIME)
IM installation time.

IMREC Field - usCodePages[16]

usCodePages[16] (USHORT)
IM supported Code Pages.

IMREC Field - LangId[2]

LangId[2] (UniChar)
IM supported Language ID.

IMREC Field - ulIMVersion

ulIMVersion (ULONG)
IM major and minor version with the following format:

Byte 1	Major version
Bytes 2-4	Minor version

IMREC Field - hwndIM

hwndIM (HWND)
IM Window handle if loaded.

IMREC Field - IMName[16]

IMName[16] (UniChar)
Name of the IM used by the GTR to identify it. This name must be unique for each IM.

IMREC Field - IMModName[64]

IMModName[64] (UniChar)
Module name (EXE file) of the IM loaded by the GTR.

IMREC Field - VIMName[64]

VIMName[64] (UniChar)
Device Name of the virtual IM Device Driver.

IMREC Field - VIMStubName[64]

VIMStubName[64] (UniChar)
Stub Module Name (usually has a SYS extension) worked with VIM.

IMREC Field - IMDescription[64]

IMDescription[64] (UniChar)
Text buffer for the IM.

IMREC Field - IMOptions[64]

IMOptions[64] (UniChar)
Text buffer for the IM.

IMREQCONV

IMREQCONV data structure.

```
typedef struct _IMREQCONV {  
    ULONG      ulSize;  
    HWND       hwndFrame;  
    ULONG      ulConvType;  
} IMREQCONV;  
  
typedef IMREQCONV *PIMREQCONV;
```

IMREQCONV Field - ulSize

ulSize (ULONG)
Size, in bytes, of this data structure.

IMREQCONV Field - hwndFrame

hwndFrame (HWND)
Frame window handle to be handled.

IMREQCONV Field - ulConvType

ulConvType (ULONG)
Conversion type. May be one of the following types:

IM_REQ_CNV_NEXT	Next candidate (= VK_CONV)
IM_REQ_CNV_PREF	Previous candidate (= VK_PREVCANDIDATE)
IM_REQ_CNV_ALL	All candidates (= VK_ALLCANDIDATE)
IM_REQ_CNV_UNDO	Undo conversion (= VK_NOCONV)

IMSTRING

IMSTRING structure.

```
typedef struct _IMSTRING {
    ULONG    ulsize;    /* Size, in bytes, of this data structure. */
    HWND     hwnd;      /* Reserved, set to NULL. */
    ULONG    ulFlag;
    BYTE     IMOut[1]; /* Consists of the following buffers: */
} IMSTRING;

typedef IMSTRING *PIMSTRING;
```

IMSTRING Field - ulsize

ulsize (ULONG)
Size, in bytes, of this data structure.

IMSTRING Field - hwnd

hwnd (HWND)
Reserved, set to NULL.

IMSTRING Field - ulFlag

ulFlag (ULONG)
Indicates the character coding scheme of the output string buffer:

Bit 0	ASCII or UNICODE characters
1	IM_CHR_MOD_ASCII ASCII(CodePage) characters
0	IM_CHR_MOD_UNICODE UNICODE characters
Bit 1-31	Reserved. Set to zero.

IMSTRING Field - IMOut[1]

IMOut[1] (BYTE)
Consists of the following buffers:

BYTE OutStr[]	Output string
BYTE OutAttr[]	Character attribute

IMWRDREG

IMWRDREG data structure.

```
typedef struct _IMWRDREG {
    ULONG      ulSize;
    HWND       hwndFrame;
    ULONG      ulFlag;
    ULONG      ulObjLen;
    PSZ        pObjBuf;
    ULONG      ulRefLen;
    PSZ        pRefBuf;
} IMWRDREG;

typedef IMWRDREG *PIMWRDREG;
```

IMWRDREG Field - ulSize

ulSize (ULONG)
Size, in bytes, of this data structure.

IMWRDREG Field - hwndFrame

hwndFrame (HWND)
Frame window handle to be processed.

IMWRDREG Field - ulFlag

ulFlag (ULONG)
Indicates the character coding scheme of Object and Reference string buffers:

Bit 0	Specify ASCII or UNICODE character.
1	IM_CHR_MOD_ASCII ASCII (CodePage) character
0	IM_CHAR_MOD_UNICODE UNICODE character
Bit 1-31	Reserved, set to zero

IMWRDREG Field - ulObjLen

ulObjLen (ULONG)

Length of the ObjectBuffer pointed to by pObjBuf.

IMWRDREG Field - pObjBuf

pObjBuf (PSZ)

Pointer to the ObjectBuffer to be registered.

For example, "International Business Machines Corporation".

IMWRDREG Field - ulRefLen

ulRefLen (ULONG)

Length of the ReferenceBuffer pointed to by pRefBuf.

IMWRDREG Field - pRefBuf

pRefBuf (PSZ)

Pointer to the ReferenceBuffer to be registered.

For example, "IBM".

STRPKT

Conversion string packet.

```
typedef struct _STRPKT {
    ULONG      ulsize; /* Size, in bytes, of this data structure. */
    HWND      hwnd; /* Reserved, set to NULL. */
    PCNVSTR    pCnvStr;
} STRPKT;

typedef STRPKT *PSTRPKT;
```

STRPKT Field - ulsize

ulsize (ULONG)

Size, in bytes, of this data structure.

STRPKT Field - hwnd

hwnd (HWND)
Reserved, set to NULL.

STRPKT Field - pCnvStr

pCnvStr ([PCNVSTR](#))
Pointer to the [CNVSTR](#) structure contained in the IM's output string.

Control Program

This chapter is an addendum to the OS/2 Warp Version 3 *Control Program Programming Reference* , which contains basic operating system functions.

The chapter includes the following sections:

- [CP Functions](#)
 - [CP Data Types](#)
-

CP Functions

This sections describes the following Control Program functions that are new or enhanced on OS/2 Warp (PowerPC Edition).

- [DosFindFirst](#)
 - [DosFindFromName](#)
 - [DosGetEnv](#)
 - [DosLoadResourceModule](#)
 - [DosQueryCp](#)
 - [DosQueryDosProperty](#)
 - [DosQueryFileInfo](#)
 - [DosQueryModFromAddr](#)
 - [DosQueryPathInfo](#)
 - [DosQueryProcessInfo](#)
 - [DosQueryThreadInfo](#)
 - [DosQueryUconvObject](#)
 - [DosQueryUnicodeCpString](#)
 - [DosReplaceModule](#)
 - [DosSelectSession](#)
 - [DosSetProcessCp](#)
 - [DosSetDosProperty](#)
 - [DosSetSession](#)
 - [DosShutdown](#)
 - [DosStopSession](#)
 - [DosTestAddr](#)
 - [DosUnwindException](#)
-

DosFindFirst

DosFindFirst - Syntax

DosFindFirst finds the first file object or group of file objects whose names match the specification. The specification can include extended attributes associated with a file or directory.

```
#define INCL_DOSFILEMGR
#include <os2.h>

PSZ      pszFileSpec; /* Address of the ASCIIZ path name of the file or subdirectory to be found. */
PHDIR    phdir;       /* Address of the handle associated with this DosFindFirst request. */
ULONG    flAttribute; /* Attribute value that determines the file objects to be searched for. */
PVOID    pfindbuf;    /* Result buffer. */
ULONG    cbBuf;        /* The length, in bytes, of pfindbuf. */
PULONG    pcFileNames; /* Pointer to the number of entries. */
ULONG    ulInfoLevel;  /* The level of file information required. */
APIRET    ulrc;        /* Return Code. */

ulrc = DosFindFirst(pszFileSpec, phdir, flAttribute,
                    pfindbuf, cbBuf, pcFileNames, ulInfoLevel);
```

DosFindFirst Parameter - pszFileSpec

pszFileSpec (PSZ) - input
Address of the ASCIIZ path name of the file or subdirectory to be found.

The name component may contain global file-name characters.

DosFindFirst Parameter - phdir

phdir (PHDIR) - in/out
Address of the handle associated with this DosFindFirst request.

The values that can be specified for the handle are shown in the following list:

HDIR_SYSTEM (0x00000001)
The system assigns the handle for standard output, which is always available to a process.

HDIR_CREATE (0xFFFFFFFF)
The system allocates and returns a handle. Upon return to the caller, *phdir* contains the handle allocated by the system.

The DosFindFirst handle is used with subsequent DosFindNext requests. Reuse of this handle in another DosFindFirst request closes the association with the previous DosFindFirst request, and opens a new association with the current DosFindFirst request.

DosFindFirst Parameter - flAttribute

flAttribute (ULONG) - input

Attribute value that determines the file objects to be searched for.

The bit values are shown in the following list:

31-14	Reserved; must be zero.
13	MUST_HAVE_ARCHIVED (0x00002000) Must have Archive bit; excludes files without the archive bit set if bit 13 is set to 1. Files may have the Archive bit set if bit 13 is set to 0.
12	MUST_HAVE_DIRECTORY (0x00001000) Must have Subdirectory bit; excludes files that are not subdirectories if bit 12 is set to 1. Files may have the Subdirectory bit set if bit 12 is set to 0.
11	Reserved; must be zero.
10	MUST_HAVE_SYSTEM (0x00000400) Must have System File bit; excludes non-system files if bit 10 is set to 1. Files may be system files if bit 10 is set to 0.
9	MUST_HAVE_HIDDEN (0x00000200) Must have Hidden File bit; excludes non-hidden files if bit 9 is set to 1. Files may be non-hidden if bit 9 is set to 0.
8	MUST_HAVE_READONLY (0x00000100) Must have Read-Only File bit; excludes writeable files if bit 8 is set to 1. Files may be read-only if bit 8 is set to 0.
7-6	Reserved; must be zero.
5	FILE_ARCHIVED (0x00000020) May have Archive bit; includes files with the Archive bit set if bit 5 is set to 1. Excludes files with the Archive bit set if bit 5 is set to 0.
4	FILE_DIRECTORY (0x00000010) May have Subdirectory bit; includes files that are subdirectories if bit 4 is set to 1. Excludes files that are subdirectories if bit 4 is set to 0.
3	Reserved; must be zero.
2	FILE_SYSTEM (0x00000004) May have System File bit; includes system files if bit 2 is set to 1. Excludes system files if bit 2 is set to 0.
1	FILE_HIDDEN (0x00000002) May have Hidden File bit; includes hidden files if bit 1 is set to 1. Excludes hidden files if bit 1 is set to 0.
0	FILE_READONLY (0x00000001) May have Read-Only File bit; includes read-only files if bit 0 is set to 1. Excludes read-only files if bit 0 is set to 0.

These bits may be set individually or in combination. For example, an attribute value of 0x00000021 (bits 5 and 0 set to 1) indicates searching for read-only files that have been archived.

Bits 8 through 13 are "Must-have" flags. These allow you to obtain files that definitely have the given attributes. For example, if the Must have Subdirectory bit is set to 1, then all returned items are subdirectories.

If a Must-have bit is set to 1 and the corresponding May-have bit is set to zero, no items are returned for that attribute.

The attribute FILE_NORMAL (0x00000000) can be used to include files with any of the above bits set.

flAttribute cannot specify the volume label. Volume labels are queried using DosQueryFSInfo.

DosFindFirst Parameter - pfindbuf

pfindbuf (PVOID) - in/out
Result buffer.

The result buffer from DosFindFirst should be less than 64Kb.

Address of the directory search structures for file object information levels 1 through 4 and 11 through 14. The structure required for *pfindbuf* is dependent on the value specified for *ulInfoLevel*. The information returned reflects the most recent call to DosClose or DosResetBuffer.

For Level 1 File Information (*ulInfoLevel* == FIL_STANDARD) :

On output, *pfindbuf* contains the [FILEFINDBUF3](#) data structure without the last two fields: *cchName* and *achName*. This is used without extended attributes (EA).

The *oNextEntryOffset* field indicates the number of bytes from the beginning of the current structure to the beginning of the next structure. When this field is zero, the last structure has been reached.

For Level 2 File Information (*ulInfoLevel* == FIL_QUERYEASIZE) :

On output, *pfindbuf* contains the [FILEFINDBUF4](#) data structure without the last two fields: *cchName* and *achName*. This is used with EAs.

The *cbList* field contains the size, in bytes, of the file's entire EA set on disk. You can use this field to calculate the maximum size of the buffer needed for level 3 file information. The size of the buffer required to hold the entire EA set is less than or equal to twice the size of the EA set on disk.

For Level 3 File Information (*ulInfoLevel* == FIL_QUERYEASFROMLIST) :

On input, *pfindbuf* contains an [EAOP2](#) data structure. *fpGEA2List* contains a pointer to a [GEA2](#) list, which defines the attribute names whose values are to be returned. Entries in the [GEA2](#) list must be aligned on a doubleword boundary. Each *oNextEntryOffset* field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry.

On output, *pfindbuf* contains a structure with a set of records, each aligned on a doubleword boundary. These records represent the directory entry and associated extended attributes (EA) for the matched file object. *pfindbuf* has the following format:

- The [EAOP2](#) data structure, with the *fpFEA2List* pointer incorrect.

The [EAOP2](#) data structure occurs only once in the *pfindbuf* buffer. The rest of these records are repeated for the remainder of the file objects found.
- A [FILEFINDBUF3](#) data structure without the last two fields: *cchName* and *achName*.
- A [FEA2LIST](#) data structure contained in and related to the [FILEFINDBUF3](#) returned.
- Length of the name string of the file object (*cbName*)
- Name of the file object matched by the input pattern (*achName*)

Even if there is not enough room to hold all of the requested information, as for return code ERROR_BUFFER_OVERFLOW, the *cbList* field of the [FEA2LIST](#) data structure is valid if there is at least enough space to hold it.

When buffer overflow occurs, *cbList* contains the size on disk of the entire EA set for the file, even if only a subset of its attributes was requested. The size of the buffer required to hold the EA set is less than or equal to twice the size of the EA set on disk. If no error occurs, *cbList* includes the pad bytes (for doubleword alignment) between [FEA2](#) structures in the list.

If a particular attribute is not attached to the object, *pfindbuf* has an [FEA2](#) structure containing the name of the attribute, and the length value is zero.

The [GEA2](#) list contained inside *pfindbuf* during a level 3 DosFindFirst and DosFindNext call is not "read-only", it is used by OS/2. When the function returns, the list is restored to its original state, but inside the function, the list is manipulated by OS/2. This is of concern to a multithreaded application, where two different threads might use the same [GEA2](#) list as input. If one thread calls DosFindFirst or DosFindNext while another thread is inside DosFindFirst or DosFindNext, the second thread will fail with a return code of ERROR_BAD_FORMAT.

For Level 4 File Information (*ulInfoLevel* == FIL_QUERYALLEAS) :

On output, same as Level 3.

For Level 11 File Information (*ulInfoLevel* == FIL_STANDARD2) :

Same as Level 1, except *pfindbuf* contains the [FILEFINDBUF13](#) data structure. [FILEFINDBUF13](#) includes *position*, which is used by DosFindFromName.

For Level 12 File Information (*ulInfoLevel* == `FIL_QUERYSIZE2`) :

Same as Level 2, except *pfindbuf* contains the `FILEFINDBUF14` data structure. `FILEFINDBUF14` includes *position* , which is used by `DosFindFromName`.

For Level 13 File Information (*ulInfoLevel* == `FIL_QUERYEASFROMLIST2`) :

Same as Level 3, except *pfindbuf* contains the `FILEFINDBUF14` data structure. `FILEFINDBUF14` includes *position* , which is used by `DosFindFromName`.

DosFindFirst Parameter - cbBuf

cbBuf (ULONG) - input

The length, in bytes, of *pfindbuf* .

DosFindFirst Parameter - pcFileNames

pcFileNames (PULONG) - in/out

Pointer to the number of entries.

Input

The address of the number of matching entries requested in *pfindbuf* .

Output

The number of entries placed into *pfindbuf* .

DosFindFirst Parameter - ulInfoLevel

ulInfoLevel (ULONG) - input

The level of file information required.

Possible values are shown in the list below:

- | | |
|----|--|
| 1 | <code>FIL_STANDARD</code>
Level 1 file information (return standard file information) |
| 2 | <code>FIL_QUERYEASIZE</code>
Level 2 file information (return full EA size) |
| 3 | <code>FIL_QUERYEASFROMLIST</code>
Level 3 file information (return requested EAs) |
| 4 | <code>FIL_QUERYALLEAS</code>
Level 4 file information (return all EAs) |
| 11 | <code>FIL_STANDARD2</code>
Level 1 file information with <i>position</i> |
| 12 | <code>FIL_QUERYEASIZE2</code>
Level 2 file information with <i>position</i> |
| 13 | <code>FIL_QUERYEASFROMLIST2</code>
Level 3 file information with <i>position</i> |
| 14 | <code>FIL_QUERYALLEAS2</code>
Level 4 file information with <i>position</i> |

The structures described in *pfindbuf* indicate the information returned for each of these levels.

Regardless of the level specified, a DosFindFirst request (and an associated DosFindNext) request on a handle returned by DosFindFirst) always includes level 1 information as part of the information that is returned; however, when level 1 information is specifically requested, and *flAttribute* specifies hidden files, system files, or subdirectory files, an inclusive search is made. That is, all normal file entries plus all entries matching any specified attributes are returned. Normal files are files without any mode bits set. They may be read from or written to.

DosFindFirst Return Value - ulrc

ulrc (APIRET) - returns
Return Code.

DosFindFirst returns one of the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
111	ERROR_BUFFER_OVERFLOW
113	ERROR_NO_MORE_SEARCH_HANDLES
206	ERROR_FILENAME_EXCED_RANGE
208	ERROR_META_EXPANSION_TOO_LONG
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT
275	ERROR_EAS_DIDNT_FIT

DosFindFirst - Parameters

pszFileSpec (PSZ) - input
Address of the ASCII path name of the file or subdirectory to be found.

The name component may contain global file-name characters.

phdir (PHDIR) - in/out
Address of the handle associated with this DosFindFirst request.

The values that can be specified for the handle are shown in the following list:

HDIR_SYSTEM (0x00000001)

The system assigns the handle for standard output, which is always available to a process.

HDIR_CREATE (0xFFFFFFFF)

The system allocates and returns a handle. Upon return to the caller, *phdir* contains the handle allocated by the system.

The DosFindFirst handle is used with subsequent DosFindNext requests. Reuse of this handle in another DosFindFirst request closes the association with the previous DosFindFirst request, and opens a new association with the current DosFindFirst request.

flAttribute (ULONG) - input
Attribute value that determines the file objects to be searched for.

The bit values are shown in the following list:

31-14 Reserved; must be zero.

13 MUST_HAVE_ARCHIVED (0x00002000)

	Must have Archive bit; excludes files without the archive bit set if bit 13 is set to 1. Files may have the Archive bit set if bit 13 is set to 0.
12	MUST_HAVE_DIRECTORY (0x00001000) Must have Subdirectory bit; excludes files that are not subdirectories if bit 12 is set to 1. Files may have the Subdirectory bit set if bit 12 is set to 0.
11	Reserved; must be zero.
10	MUST_HAVE_SYSTEM (0x00000400) Must have System File bit; excludes non-system files if bit 10 is set to 1. Files may be system files if bit 10 is set to 0.
9	MUST_HAVE_HIDDEN (0x00000200) Must have Hidden File bit; excludes non-hidden files if bit 9 is set to 1. Files may be non-hidden if bit 9 is set to 0.
8	MUST_HAVE_READONLY (0x00000100) Must have Read-Only File bit; excludes writeable files if bit 8 is set to 1. Files may be read-only if bit 8 is set to 0.
7-6	Reserved; must be zero.
5	FILE_ARCHIVED (0x00000020) May have Archive bit; includes files with the Archive bit set if bit 5 is set to 1. Excludes files with the Archive bit set if bit 5 is set to 0.
4	FILE_DIRECTORY (0x00000010) May have Subdirectory bit; includes files that are subdirectories if bit 4 is set to 1. Excludes files that are subdirectories if bit 4 is set to 0.
3	Reserved; must be zero.
2	FILE_SYSTEM (0x00000004) May have System File bit; includes system files if bit 2 is set to 1. Excludes system files if bit 2 is set to 0.
1	FILE_HIDDEN (0x00000002) May have Hidden File bit; includes hidden files if bit 1 is set to 1. Excludes hidden files if bit 1 is set to 0.
0	FILE_READONLY (0x00000001) May have Read-Only File bit; includes read-only files if bit 0 is set to 1. Excludes read-only files if bit 0 is set to 0.

These bits may be set individually or in combination. For example, an attribute value of 0x00000021 (bits 5 and 0 set to 1) indicates searching for read-only files that have been archived.

Bits 8 through 13 are "Must-have" flags. These allow you to obtain files that definitely have the given attributes. For example, if the Must have Subdirectory bit is set to 1, then all returned items are subdirectories.

If a Must-have bit is set to 1 and the corresponding May-have bit is set to zero, no items are returned for that attribute.

The attribute FILE_NORMAL (0x00000000) can be used to include files with any of the above bits set.

flAttribute cannot specify the volume label. Volume labels are queried using DosQueryFSInfo.

pfindbuf (PVOID) - in/out
Result buffer.

The result buffer from DosFindFirst should be less than 64Kb.

Address of the directory search structures for file object information levels 1 through 4 and 11 through 14. The structure required for *pfindbuf* is dependent on the value specified for *ulInfoLevel*. The information returned reflects the most recent call to DosClose or DosResetBuffer.

For Level 1 File Information (*ulInfoLevel* == FIL_STANDARD) :

On output, *pfindbuf* contains the [FILEFINDBUF3](#) data structure without the last two fields: *cchName* and *achName*. This is used without extended attributes (EA).

The *oNextEntryOffset* field indicates the number of bytes from the beginning of the current structure to the beginning of the next structure. When this field is zero, the last structure has been reached.

For Level 2 File Information (*ulInfoLevel* == FIL_QUERYEASIZE) :

On output, *pfindbuf* contains the [FILEFINDBUF4](#) data structure without the last two fields: *cchName* and *achName*. This is used with EAs.

The *cbList* field contains the size, in bytes, of the file's entire EA set on disk. You can use this field to calculate the maximum size of the buffer needed for level 3 file information. The size of the buffer required to hold the entire EA set is less than or equal to twice the size of the EA set on disk.

For Level 3 File Information (*ulInfoLevel* == `FIL_QUERYEASFROMLIST`) :

On input, *pfindbuf* contains an [EAOP2](#) data structure. *fpGEA2List* contains a pointer to a [GEA2](#) list, which defines the attribute names whose values are to be returned. Entries in the [GEA2](#) list must be aligned on a doubleword boundary. Each *oNextEntryOffset* field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry.

On output, *pfindbuf* contains a structure with a set of records, each aligned on a doubleword boundary. These records represent the directory entry and associated extended attributes (EA) for the matched file object. *pfindbuf* has the following format:

- The [EAOP2](#) data structure, with the *fpFEA2List* pointer incorrect.

The [EAOP2](#) data structure occurs only once in the *pfindbuf* buffer. The rest of these records are repeated for the remainder of the file objects found.

- A [FILEFINDBUF3](#) data structure without the last two fields: *cchName* and *achName* .
- A [FEA2LIST](#) data structure contained in and related to the [FILEFINDBUF3](#) returned.
- Length of the name string of the file object (*cbName*)
- Name of the file object matched by the input pattern (*achName*)

Even if there is not enough room to hold all of the requested information, as for return code `ERROR_BUFFER_OVERFLOW`, the *cbList* field of the [FEA2LIST](#) data structure is valid if there is at least enough space to hold it.

When buffer overflow occurs, *cbList* contains the size on disk of the entire EA set for the file, even if only a subset of its attributes was requested. The size of the buffer required to hold the EA set is less than or equal to twice the size of the EA set on disk. If no error occurs, *cbList* includes the pad bytes (for doubleword alignment) between [FEA2](#) structures in the list.

If a particular attribute is not attached to the object, *pfindbuf* has an [FEA2](#) structure containing the name of the attribute, and the length value is zero.

The [GEA2](#) list contained inside *pfindbuf* during a level 3 `DosFindFirst` and `DosFindNext` call is not "read-only", it is used by OS/2. When the function returns, the list is restored to its original state, but inside the function, the list is manipulated by OS/2. This is of concern to a multithreaded application, where two different threads might use the same [GEA2](#) list as input. If one thread calls `DosFindFirst` or `DosFindNext` while another thread is inside `DosFindFirst` or `DosFindNext`, the second thread will fail with a return code of `ERROR_BAD_FORMAT`.

For Level 4 File Information (*ulInfoLevel* == `FIL_QUERYALLEAS`) :

On output, same as Level 3.

For Level 11 File Information (*ulInfoLevel* == `FIL_STANDARD2`) :

Same as Level 1, except *pfindbuf* contains the [FILEFINDBUF13](#) data structure. [FILEFINDBUF13](#) includes *position* , which is used by `DosFindFromName`.

For Level 12 File Information (*ulInfoLevel* == `FIL_QUERYSIZE2`) :

Same as Level 2, except *pfindbuf* contains the [FILEFINDBUF14](#) data structure. [FILEFINDBUF14](#) includes *position* , which is used by `DosFindFromName`.

For Level 13 File Information (*ulInfoLevel* == `FIL_QUERYEASFROMLIST2`) :

Same as Level 3, except *pfindbuf* contains the [FILEFINDBUF14](#) data structure. [FILEFINDBUF14](#) includes *position* , which is used by `DosFindFromName`.

cbBuf (ULONG) - input

The length, in bytes, of *pfindbuf* .

pcFileNames (PULONG) - in/out

Pointer to the number of entries.

Input

The address of the number of matching entries requested in *pfindbuf* .

Output

The number of entries placed into *pfindbuf* .

ulInfoLevel (ULONG) - input

The level of file information required.

Possible values are shown in the list below:

1	FIL_STANDARD Level 1 file information (return standard file information)
2	FIL_QUERYEASIZE Level 2 file information (return full EA size)
3	FIL_QUERYEASFROMLIST Level 3 file information (return requested EAs)
4	FIL_QUERYALLEAS Level 4 file information (return all EAs)
11	FIL_STANDARD2 Level 1 file information with <i>position</i>
12	FIL_QUERYEASIZE2 Level 2 file information with <i>position</i>
13	FIL_QUERYEASFROMLIST2 Level 3 file information with <i>position</i>
14	FIL_QUERYALLEAS2 Level 4 file information with <i>position</i>

The structures described in *pfindbuf* indicate the information returned for each of these levels.

Regardless of the level specified, a DosFindFirst request (and an associated DosFindNext) request on a handle returned by DosFindFirst) always includes level 1 information as part of the information that is returned; however, when level 1 information is specifically requested, and *flAttribute* specifies hidden files, system files, or subdirectory files, an inclusive search is made. That is, all normal file entries plus all entries matching any specified attributes are returned. Normal files are files without any mode bits set. They may be read from or written to.

ulrc (APIRET) - returns
Return Code.

DosFindFirst returns one of the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
111	ERROR_BUFFER_OVERFLOW
113	ERROR_NO_MORE_SEARCH_HANDLES
206	ERROR_FILENAME_EXCED_RANGE
208	ERROR_META_EXPANSION_TOO_LONG
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT
275	ERROR_EAS_DIDNT_FIT

DosFindFirst - Remarks

The Result Buffer from DosFindFirst should be less than 64Kb.

DosFindFirst returns directory entries (up to the number requested in *pcFileNames*) and extended-attribute information for as many files or subdirectories whose names, attributes, and extended attributes match the specification, and whose information fits in *pfindbuf* . On output, *pcFileNames* contains the actual number of directory entries returned.

The file name pointed to by *pszFileSpec* can contain global file-name characters.

DosFindNext uses the directory handle associated with DosFindFirst to continue the search started by the DosFindFirst request.

Any non-zero return code, except ERROR_EAS_DIDNT_FIT, indicates that no handle has been allocated. This includes such non-error

indicators as `ERROR_NO_MORE_FILES`.

For `ERROR_EAS_DIDNT_FIT`, a search handle is returned, and a subsequent call to `DosFindNext` gets the next matching entry in the directory. You can use [DosQueryPathInfo](#) to retrieve the extended attributes (EAs) for the matching entry by using the same EA arguments used for the `DosFindFirst` call, and the name that was returned by `DosFindFirst`.

For `ERROR_EAS_DIDNT_FIT`, only information for the first matching entry is returned. This entry is the one whose extended attributes did not fit in the buffer. The information returned is in the format of that returned for information level 2. No further entries are returned in the buffer, even if they could fit in the remaining space.

The [GEA2](#) list contained inside *pfindbuf* during a level 3 `DosFindFirst` and `DosFindNext` call is not "read-only", it is used by OS/2. When the function returns, the list is restored to its original state, but inside the function, the list is manipulated by OS/2. This is of concern to a multithreaded application, where two different threads might use the same [GEA2](#) list as input. If one thread calls `DosFindFirst` or `DosFindNext` while another thread is inside `DosFindFirst` or `DosFindNext`, the second thread will fail with a return code of `ERROR_BAD_FORMAT`.

DosFindFirst - Related Functions

- `DosClose`
- `DosFindClose`
- `DosFindNext`
- [DosQueryFileInfo](#)
- [DosQueryPathInfo](#)
- `DosQuerySysInfo`
- `DosResetBuffer`
- `DosSearchPath`
- `DosSetFileInfo`
- `DosSetPathInfo`

DosFindFirst - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

DosFindFromName

DosFindFromName - Syntax

`DosFindFromName` finds the next set of entries that match the criteria specified in a previous call to [DosFindFirst](#), continuing a directory search from a given position.

```
#include <os2.h>
```

```
HDIR      hDir;          /* The open directory handle associated with this search request. */
PVOID      pfindbuf;     /* Pointer to the directory search structures. */
ULONG      cbBuf;        /* Length, in bytes, the buffer returned in pfindbuf. */
PULONG     pcFileNames;  /* Number of entries in the buffer returned in pfindbuf. */
```

```
ULONG      ulPosition; /* Starting position for search. */
PVOID      pszFileSpec; /* File name to start with. */
APIRET     rc; /* Return code. */
```

```
rc = DosFindFromName(hDir, pfindbuf, cbBuf,  
    pcFileNames, ulPosition, pszFileSpec);
```

DosFindFromName Parameter - hDir

hDir (HDIR) - input

The open directory handle associated with this search request.

This handle is the same one used by [DosFindFirst](#).

DosFindFromName Parameter - pfindbuf

pfindbuf (PVOID) - in/out

Pointer to the directory search structures.

DosFindFromName Parameter - cbBuf

cbBuf (ULONG) - input

Length, in bytes, the buffer returned in *pfindbuf* .

DosFindFromName Parameter - pcFileNames

pcFileNames (PULONG) - in/out

Number of entries in the buffer returned in *pfindbuf* .

DosFindFromName Parameter - ulPosition

ulPosition (ULONG) - input

Starting position for search.

DosFindFromName Parameter - pszFileSpec

pszFileSpec (PVOID) - input
File name to start with.

pszFileSpec is in either Unicode or ASCII depending on how the search handle was created. If the search handle was opened by *DosFindFirstUni*, then *pszFileSpec* is in Unicode; otherwise, it is in ASCII.

DosFindFromName Return Value - rc

rc (APIRET) - returns
Return code.

DosFindFromName returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
275	ERROR_EAS_DIDNT_FIT

DosFindFromName - Parameters

hDir (HDIR) - input
The open directory handle associated with this search request.

This handle is the same one used by [DosFindFirst](#).

pfindbuf (PVOID) - in/out
Pointer to the directory search structures.

cbBuf (ULONG) - input
Length, in bytes, the buffer returned in *pfindbuf*.

pcFileNames (PULONG) - in/out
Number of entries in the buffer returned in *pfindbuf*.

ulPosition (ULONG) - input
Starting position for search.

pszFileSpec (PVOID) - input
File name to start with.

pszFileSpec is in either Unicode or ASCII depending on how the search handle was created. If the search handle was opened by *DosFindFirstUni*, then *pszFileSpec* is in Unicode; otherwise, it is in ASCII.

rc (APIRET) - returns
Return code.

DosFindFromName returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
275	ERROR_EAS_DIDNT_FIT

DosFindFromName - Remarks

DosFindFromName allows you to perform a find using a name. The search would start at the beginning of the directory, but it would only return data from *pszFileSpec* on. This reduces the load on the network.

DosFindFromName - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DosGetEnv

DosGetEnv - Syntax

DosGetEnv returns the process' environment string in the specified buffer.

```
#include <os2.h>

PULONG   BufferLength;
PVOID     Buffer;
APIRET    rc;

rc = DosGetEnv(BufferLength, Buffer);
```

DosGetEnv Parameter - BufferLength

BufferLength (PULONG) - in/out

On input, the length of the buffer provided by the caller. Output, the total number of bytes placed in the provided buffer. If the caller's buffer is too small, this will be set to the required size.

DosGetEnv Parameter - Buffer

Buffer (PVOID) - output

Address of the buffer where the data is returned.

DosGetEnv Return Value - rc

rc (APIRET) - returns

Return codes.

0	
111	NO_ERROR
	ERROR_BUFFER_OVERFLOW

DosGetEnv - Parameters

BufferLength (PULONG) - in/out

On input, the length of the buffer provided by the caller. Output, the total number of bytes placed in the provided buffer. If the caller's buffer is too small, this will be set to the required size.

Buffer (PVOID) - output

Address of the buffer where the data is returned.

rc (APIRET) - returns

Return codes.

0	
111	NO_ERROR
	ERROR_BUFFER_OVERFLOW

DosGetEnv - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

DosGetLocaleModule

DosGetLocaleModule - Syntax

DosGetLocaleModule loads a resource based on the locale currently in use or the system language. It returns the handle to the module.

```
#include <os2.h>

PHMODULE    phmod;    /* An HMODULE is returned for the resource module. */
PSZ         pszModname; /* The address of the resource name. */
PSZ         pszLocname; /* The address of the resource name. */
APIRET      ulrc;     /* Return code. */

ulrc = DosGetLocaleModule(phmod, pszModname,
                          pszLocname);
```

DosGetLocaleModule Parameter - phmod

phmod (PHMODULE) - output
An HMODULE is returned for the resource module.

DosGetLocaleModule Parameter - pszModname

pszModname (PSZ) - input
The address of the resource name.

The address of a zero-terminated ASCII string that contains the name to be used in creating the resource name. This is a simple string of 1-6 characters.

DosGetLocaleModule Parameter - pszLocname

pszLocname (PSZ) - input
The address of the resource name.

The address of a zero-terminated ASCII string that contains the name of the locale to be used. If NULL or a null string is provided, the system will use the current locale.

DosGetLocaleModule Return Value - ulrc

ulrc (APIRET) - returns
Return code.

DosGetLocaleModule returns one of the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
11	ERROR_BAD_FORMAT
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
123	ERROR_INVALID_NAME
127	ERROR_PROC_NOT_FOUND
180	ERROR_INVALID_SEGMENT_NUMBER
182	ERROR_INVALID_ORDINAL
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
194	ERROR_ITERATED_DATA_EXCEEDS_64K
195	ERROR_INVALID_MINALLOCSIZE
196	ERROR_DYNLINK_FROM_INVALID_RING
198	ERROR_INVALID_SEGDPL
199	ERROR_AUTODATASEG_EXCEEDS_64K
201	ERROR_RELOC_SRC_CHAIN_EXCEEDS_SEGLIMIT
206	ERROR_FILENAME_EXCED_RANGE
295	ERROR_INIT_ROUTINE_FAILED

DosGetLocaleModule - Parameters

phmod ([PHMODULE](#)) - output
An HMODULE is returned for the resource module.

pszModname (PSZ) - input
The address of the resource name.

The address of a zero-terminated ASCII string that contains the name to be used in creating the resource name. This is a simple string of 1-6 characters.

pszLocname (PSZ) - input
The address of the resource name.

The address of a zero-terminated ASCII string that contains the name of the locale to be used. If NULL or a null string is provided, the system will use the current locale.

ulrc (APIRET) - returns
Return code.

DosGetLocaleModule returns one of the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
11	ERROR_BAD_FORMAT
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED

95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
123	ERROR_INVALID_NAME
127	ERROR_PROC_NOT_FOUND
180	ERROR_INVALID_SEGMENT_NUMBER
182	ERROR_INVALID_ORDINAL
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
194	ERROR_ITERATED_DATA_EXCEEDS_64K
195	ERROR_INVALID_MINALLOCSIZE
196	ERROR_DYNLINK_FROM_INVALID_RING
198	ERROR_INVALID_SEGDPL
199	ERROR_AUTODATASEG_EXCEEDS_64K
201	ERROR_RELOCSRC_CHAIN_EXCEEDS_SEGLIMIT
206	ERROR_FILENAME_EXCED_RANGE
295	ERROR_INIT_ROUTINE_FAILED

DosGetLocaleModule - Remarks

DosGetLocaleModule is used to facilitate the location and loading of load modules that require a two-character ISO standard language identifier as the first two characters of the dynamic link module name. This function will determine the language identifier if it is not supplied by the user.

If the language of the current or specified locale is not available, DosGetLocaleModule will attempt to find an appropriate substitute such as using the system language.

DosGetLocaleModule can be used to return a pointer to the [HMODULE](#). This pointer can then be used by DosGetResource to load a resource object.

DosGetLocaleModule - Related Functions

- [DosExecPgm](#)
- [DosFreeModule](#)
- [DosLoadModule](#)
- [DosLoadResourceModule](#)
- [DosQueryModuleName](#)
- [DosQueryProcAddr](#)

DosGetLocaleModule - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

DosLoadResourceModule

DosLoadResourceModule - Syntax

DosLoadModule loads a dynamic link or executable module, and returns a handle for the module.

```
#define INCL_DOSMODULEMGR
#include <os2.h>

PSZ      pszName;      /* The address of a buffer into which the name of an object that contributed to the fa
ULONG    cbName;       /* The length, in bytes, of the buffer described by pszName. */
PSZ      pszModname;   /* The address of an ASCIIZ name string that contains the dynamic link or executable m
PHMODULE phmod;        /* Pointer to an HMODULE in which the handle for the dynamic link or executable module
APIRET   ulrc;         /* Return Code. */

ulrc = DosLoadResourceModule(pszName, cbName,
                             pszModname, phmod);
```

DosLoadResourceModule Parameter - pszName

pszName (PSZ) - output

The address of a buffer into which the name of an object that contributed to the failure of DosLoadResourceModule is to be placed.

The name of the object is usually the name of a dynamic link library that either could not be found or could not be loaded.

DosLoadResourceModule Parameter - cbName

cbName (ULONG) - input

The length, in bytes, of the buffer described by *pszName*.

DosLoadResourceModule Parameter - pszModname

pszModname (PSZ) - input

The address of an ASCIIZ name string that contains the dynamic link or executable module name.

The file-name extension used for dynamic link libraries is .DLL.

When a request is made to load a module and a fully-qualified path is specified, the system loads that module, if it exists. If a fully-qualified path is *not* specified, the system checks if the module is already loaded. If it is loaded, that module is the one that is used; otherwise, the system searches the paths in the "CONFIG.SYS" file and uses the first instance of the specified module it finds. If the current directory is not specified in the path, the system does not check that directory to see if a different version exists. Consequently, if two processes started from different directories use the same module, but different versions of that module exist in both directories, the version of the module loaded by the first process is the one used by both processes.

DosLoadResourceModule Parameter - phmod

phmod (PHMODULE) - output
Pointer to an HMODULE in which the handle for the dynamic link or executable module is returned.

DosLoadResourceModule Return Value - ulrc

ulrc (APIRET) - returns
Return Code.

DosLoadModule returns one of the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
11	ERROR_BAD_FORMAT
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
123	ERROR_INVALID_NAME
127	ERROR_PROC_NOT_FOUND
180	ERROR_INVALID_SEGMENT_NUMBER
182	ERROR_INVALID_ORDINAL
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID
194	ERROR_ITERATED_DATA_EXCEEDS_64K
195	ERROR_INVALID_MINALLOCSIZE
196	ERROR_DYNLINK_FROM_INVALID_RING
198	ERROR_INVALID_SEGDPL
199	ERROR_AUTODATASEG_EXCEEDS_64K
201	ERROR_RELOCSRC_CHAIN_EXCEEDS_SEGLIMIT
206	ERROR_FILENAME_EXCED_RANGE
295	ERROR_INIT_ROUTINE_FAILED

DosLoadResourceModule - Parameters

pszName (PSZ) - output
The address of a buffer into which the name of an object that contributed to the failure of DosLoadResourceModule is to be placed.

The name of the object is usually the name of a dynamic link library that either could not be found or could not be loaded.

cbName (ULONG) - input
The length, in bytes, of the buffer described by *pszName* .

pszModname (PSZ) - input
The address of an ASCIIZ name string that contains the dynamic link or executable module name.

The file-name extension used for dynamic link libraries is .DLL.

When a request is made to load a module and a fully-qualified path is specified, the system loads that module, if it exists. If a fully-qualified path is *not* specified, the system checks if the module is already loaded. If it is loaded, that module is the one that is used; otherwise, the system searches the paths in the "CONFIG.SYS" file and uses the first instance of the specified module it finds. If the current directory is not specified in the path, the system does not check that directory to see if a different version exists. Consequently, if two processes started from different directories use the same module, but different versions of that module exist in

both directories, the version of the module loaded by the first process is the one used by both processes.

phmod (PHMODULE) - output

Pointer to an HMODULE in which the handle for the dynamic link or executable module is returned.

ulrc (APIRET) - returns

Return Code.

DosLoadModule returns one of the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
11	ERROR_BAD_FORMAT
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
123	ERROR_INVALID_NAME
127	ERROR_PROC_NOT_FOUND
180	ERROR_INVALID_SEGMENT_NUMBER
182	ERROR_INVALID_ORDINAL
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID
194	ERROR_ITERATED_DATA_EXCEEDS_64K
195	ERROR_INVALID_MINALLOCSIZE
196	ERROR_DYNLINK_FROM_INVALID_RING
198	ERROR_INVALID_SEGDPL
199	ERROR_AUTODATASEG_EXCEEDS_64K
201	ERROR_RELOCSRC_CHAIN_EXCEEDS_SEGLIMIT
206	ERROR_FILENAME_EXCED_RANGE
295	ERROR_INIT_ROUTINE_FAILED

DosLoadResourceModule - Remarks

DosLoadResourceModule loads a dynamic link or executable module, and returns a handle for the module.

If the file is an OS/2 dynamic link or executable module, then the module is loaded, and a handle is returned. The returned handle is used for freeing the dynamic link or executable module, getting procedure addresses, and getting the fully qualified file name.

DosLoadResourceModule cannot be issued from ring 2 if the dynamic library routine has an initialization routine, or the process will be terminated.

If the module has an initialization routine that is in an object that has IOPL indicated, any process attempting to use the module will cause a general protection fault, and will be terminated.

DosLoadResourceModule - Related Functions

- DosExecPgm
- DosFreeModule
- DosQueryModuleName
- DosQueryProcAddr

DosLoadResourceModule - Example Code

This example loads the dynamic link module "DISPLAY.DLL," queries its address and type, and finally frees it.

```
#define INCL_DOSMODULEMGR      /* Module Manager values */
#define INCL_DOSERRORS        /* Error values */
#include <os2.h>
#include <stdio.h>

int main(VOID) {

    PSZ      ModuleName      = "C:\\OS2\\DLL\\DISPLAY.DLL"; /* Name of module */
    UCHAR    LoadError[256] = ""; /* Area for Load failure information */
    HMODULE   ModuleHandle    = NULLHANDLE; /* Module handle */
    PFN       ModuleAddr      = 0; /* Pointer to a system function */
    ULONG     ModuleType       = 0; /* Module type */
    APIRET    rc              = NO_ERROR; /* Return code */

    rc = DosLoadModule(LoadError, /* Failure information buffer */
                      sizeof(LoadError), /* Size of buffer */
                      ModuleName, /* Module to load */
                      &ModuleHandle); /* Module handle returned */

    if (rc != NO_ERROR) {
        printf("DosLoadModule error: return code = %u\n", rc);
        return 1;
    } else {
        printf("Module %s loaded.\n", ModuleName);
    } /* endif */

    rc = DosQueryProcAddr(ModuleHandle, /* Handle to module */
                          1L, /* No ProcName specified */
                          NULL, /* ProcName (not specified) */
                          &ModuleAddr); /* Address returned */

    if (rc != NO_ERROR) {
        printf("DosQueryProcAddr error: return code = %u\n", rc);
        return 1;
    } else printf("Address of module is 0x%x.\n", ModuleAddr);

    rc = DosQueryProcType(ModuleHandle, /* Handle to module */
                          1L, /* Indicate no ProcName given */
                          NULL, /* ProcName (not specified) */
                          &ModuleType); /* Type 0=16-bit 1=32-bit */

    if (rc != NO_ERROR) {
        printf("DosQueryProcType error: return code = %u\n", rc);
        return 1;
    } else printf("This is a %s module.\n", (ModuleType ? "32-bit" : "16-bit"));

    rc = DosFreeModule(ModuleHandle);
    if (rc != NO_ERROR) {
        printf("DosFreeModule error: return code = %u\n", rc);
        return 1;
    } else printf("Module %s freed.\n", ModuleName);

    return NO_ERROR;
}
```

DosLoadResourceModule - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DosQueryCp

DosQueryCp - Syntax

This function queries the country code of the current OS/2 process.

```
#include <os2.h>

PULONG      CountryCode;
APIRET      rc;

rc = DosQueryCp(CountryCode);
```

DosQueryCp Parameter - CountryCode

CountryCode (PULONG) - output
The returned country code of the current process.

DosQueryCp Return Value - rc

rc (APIRET) - returns
Return codes.

0	NO_ERROR
---	----------

DosQueryCp - Parameters

CountryCode (PULONG) - output
The returned country code of the current process.

rc (APIRET) - returns
Return codes.

0	NO_ERROR
---	----------

DosQueryCp - Remarks

DosQueryCp queries the country code for the current process.

The process country code previously set by [DosSetProcessCp](#) or inherited by the process is returned in a ULONG.

DosQueryCp - Example Code

```
#define INCL_DOSNLS    /* National Language Support values */
#include <os2.h>
#include <stdio.h>

ULONG  CountryCode;
APIRET rc;                                /* Return code */

rc = DosQueryCp( &CountryCode );

if (rc != 0)
{
    printf("DosQueryCp error: return code = %ld",rc);
    return;
}
```

DosQueryCp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DosQueryDosProperty

DosQueryDosProperty - Syntax

DosQueryDosProperty allows an OS/2 session to query the setting of a DOS property.

```
#include <os2.h>

SGID      SGID;
PSZ        PszName;
ULONG      cb;
PSZ        pch;
APIRET     rc;

rc = DosQueryDosProperty(SGID, PszName, cb,
    pch);
```

DosQueryDosProperty Parameter - SGID

SGID ([SGID](#)) - input
Session identification.

DosQueryDosProperty Parameter - PszName

PszName (PSZ) - input
Property name.

DosQueryDosProperty Parameter - cb

cb (ULONG) - output
Buffer length.

DosQueryDosProperty Parameter - pch

pch (PSZ) - input
Property value.

DosQueryDosProperty Return Value - rc

rc (APIRET) - returns
Return codes.

0

NO_ERROR

DosQueryDosProperty - Parameters

SGID ([SGID](#)) - input
Session identification.

PszName (PSZ) - input
Property name.

cb (ULONG) - output
Buffer length.

pch (PSZ) - input
Property value.

rc (APIRET) - returns
Return codes.

0
NO_ERROR

DosQueryDosProperty - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

DosQueryFileInfo

DosQueryFileInfo - Syntax

DosQueryFileInfo gets file information.

```
#define INCL_DOSFILEMGR
#include <os2.h>

HFILE      hf;          /* The file handle. */
ULONG      ulInfoLevel; /* Level of file information required. */
PVOID      pInfo;       /* Address of the storage area where the system returns the requested level of file info. */
ULONG      cbInfoBuf;    /* The length, in bytes, of pInfo. */
APIRET     ulrc;         /* Return Code. */

ulrc = DosQueryFileInfo(hf, ulInfoLevel, pInfo,
                        cbInfoBuf);
```

DosQueryFileInfo Parameter - hf

hf (HFILE) - input
The file handle.

DosQueryFileInfo Parameter - ulInfoLevel

ulInfoLevel (ULONG) - input
Level of file information required.

A value of 1, 2, 3, or 4 can be specified, as follows:

- | | |
|---|---|
| 1 | FIL_STANDARD
Level 1 file information. |
| 2 | FIL_QUERYEASIZE
Level 2 file information. |
| 3 | FIL_QUERYEASFROMLIST
Level 3 file information. |
| 4 | FIL_QUERYEASALL
Level-4 file information. |

The structures described in *pInfo* indicate the information returned for each of these levels.

DosQueryFileInfo Parameter - pInfo

pInfo (PVOID) - output
Address of the storage area where the system returns the requested level of file information.

File information, where applicable, is at least as accurate as the most recent DosClose, DosResetBuffer, DosSetFileInfo, or DosSetPathInfo.

Level 1 File Information (*ulInfoLevel* == FIL_STANDARD)
pInfo contains the [FILESTATUS3](#) data structure, in which file information is returned.

Level 2 File Information (*ulInfoLevel* == FIL_QUERYEASIZE)
pInfo contains the [FILESTATUS4](#) data structure. This is similar to the Level 1 structure, with the addition of the *cbList* field after the *attrFile* field.

The *cbList* field is an unsigned ULONG. On output, this field contains the size, in bytes, of the file's entire extended attribute (EA) set on disk. You can use this value to calculate the size of the buffer required to hold the EA information returned when a value of 3 is specified for *ulInfoLevel*. The buffer size is less than or equal to twice the size of the file's entire EA set on disk.

Level 3 File Information (*ulInfoLevel* == FIL_QUERYEASFROMLIST)

Input	<i>pInfo</i> contains an EAOP2 data structure. <i>fpGEA2List</i> points to a GEA2 list defining the attribute names whose values are returned. The GEA2 data structures must be aligned on a doubleword boundary. Each <i>oNextEntryOffset</i> field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry in the GEA2 list. The <i>oNextEntryOffset</i> field in the last entry of the GEA2 list must be zero. <i>fpFEA2List</i> points to a data area where the relevant FEA2 list is returned. The length field of this FEA2 list is valid, giving the size of the FEA2 list buffer. <i>oError</i> is ignored.
-------	---

Output	<i>pInfo</i> is unchanged. The buffer pointed to by <i>fpFEA2List</i> is filled in with the returned information. If the buffer that <i>fpFEA2List</i> points to is not large enough to hold the returned information (the return code is ERROR_BUFFER_OVERFLOW), <i>cbList</i> is still valid, assuming there is at least enough space for it. Its value is the size of the entire EA set on disk for the file, even though only a subset of attributes was requested.
--------	---

DosQueryFileInfo Parameter - cbInfoBuf

cbInfoBuf (ULONG) - input
The length, in bytes, of *pInfo* .

DosQueryFileInfo Return Value - ulrc

ulrc (APIRET) - returns
Return Code.

DosQueryFileInfo returns one of the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
130	ERROR_DIRECT_ACCESS_HANDLE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

DosQueryFileInfo - Parameters

hf (HFILE) - input
The file handle.

ulInfoLevel (ULONG) - input
Level of file information required.

A value of 1, 2, 3, or 4 can be specified, as follows:

1	FIL_STANDARD Level 1 file information.
2	FIL_QUERYEASIZE Level 2 file information.
3	FIL_QUERYEASFROMLIST Level 3 file information.
4	FIL_QUERYEASALL Level-4 file information.

The structures described in *pInfo* indicate the information returned for each of these levels.

pInfo (PVOID) - output
Address of the storage area where the system returns the requested level of file information.

File information, where applicable, is at least as accurate as the most recent DosClose, DosResetBuffer, DosSetFileInfo, or DosSetPathInfo.

Level 1 File Information (*ulInfoLevel* == FIL_STANDARD)
pInfo contains the [FILESTATUS3](#) data structure, in which file information is returned.

Level 2 File Information (*ulInfoLevel* == FIL_QUERYEASIZE)
pInfo contains the [FILESTATUS4](#) data structure. This is similar to the Level 1 structure, with the addition of the *cbList* field after the *attrFile* field.

The *cbList* field is an unsigned ULONG. On output, this field contains the size, in bytes, of the file's entire extended attribute (EA) set on disk. You can use this value to calculate the size of the buffer required to hold the

EA information returned when a value of 3 is specified for *ullInfoLevel* . The buffer size is less than or equal to twice the size of the file's entire EA set on disk.

Level 3 File Information (*ullInfoLevel* == FIL_QUERYEASFROMLIST)

Input	<i>pInfo</i> contains an EAOP2 data structure. <i>fpGEA2List</i> points to a GEA2 list defining the attribute names whose values are returned. The GEA2 data structures must be aligned on a doubleword boundary. Each <i>oNextEntryOffset</i> field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry in the GEA2 list. The <i>oNextEntryOffset</i> field in the last entry of the GEA2 list must be zero. <i>fpFEA2List</i> points to a data area where the relevant FEA2 list is returned. The length field of this FEA2 list is valid, giving the size of the FEA2 list buffer. <i>oError</i> is ignored.
Output	<i>pInfo</i> is unchanged. The buffer pointed to by <i>fpFEA2List</i> is filled in with the returned information. If the buffer that <i>fpFEA2List</i> points to is not large enough to hold the returned information (the return code is ERROR_BUFFER_OVERFLOW), <i>cbList</i> is still valid, assuming there is at least enough space for it. Its value is the size of the entire EA set on disk for the file, even though only a subset of attributes was requested.

cbInfoBuf (ULONG) - input
The length, in bytes, of *pInfo* .

ulrc (APIRET) - returns
Return Code.

DosQueryFileInfo returns one of the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
130	ERROR_DIRECT_ACCESS_HANDLE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

DosQueryFileInfo - Remarks

In the FAT file system, only date and time information contained in level-1 file information can be modified. Zero is returned for the creation and access dates and times.

To return information contained in any of the file information levels, DosQueryFileInfo must be able to read the open file. DosQueryFileInfo works only when the file is opened for read access, with a deny-write sharing mode specified for access by other processes. If another process that has specified conflicting sharing and access modes has already opened the file, any call to DosOpen will fail.

DosEnumAttribute returns the lengths of extended attributes. This information can be used to calculate what size *pInfo* needs to be to hold full-extended-attribute (FEA) information returned by DosQueryFileInfo when Level 3 is specified. The size of the buffer is calculated as follows:

Four bytes (for *oNextEntryOffset*) +
One byte (for *fEA* +
One byte (for *cbName*) +
Two bytes (for *cbValue*) +
Value of *cbName* (for the name of the EA) +
One byte (for terminating NULL in *cbName*) +
Value of *cbValue* (for the value of the EA)

DosQueryFileInfo - Related Functions

- DosClose
- DosEnumAttribute
- DosOpen
- [DosQueryPathInfo](#)
- DosResetBuffer
- DosSetFileInfo
- DosSetFileSize
- DosSetPathInfo

DosQueryFileInfo - Example Code

This example obtains the information of the file "CONFIG.SYS."

```
#define INCL_DOSFILEMGR    /* File Manager values */
#define INCL_DOSERRORS    /* DOS error values */
#include <os2.h>
#include <stdio.h>

int main(VOID) {
    UCHAR    uchFileName[80] = "C:\\CONFIG.SYS"; /* File to manipulate */
    FILESTATUS3 fsts3ConfigInfo = {{0}}; /* Buffer for file information */
    ULONG    ulBufSize = sizeof(FILESTATUS3); /* Size of above buffer */
    HFILE    hfConfig = 0; /* Handle for Config file */
    ULONG    ulOpenAction = 0; /* Action taken by DosOpen */
    APIRET    rc = NO_ERROR; /* Return code */

    rc = DosOpen(uchFileName, /* File to open (path and name) */
                 &hfConfig, /* File handle returned */
                 &ulOpenAction, /* Action taken by DosOpen */
                 0L, 0L, /* Primary allocation and attributes (ignored) */
                 OPEN_ACTION_FAIL_IF_NEW |
                 OPEN_ACTION_OPEN_IF_EXISTS, /* Open an existing file only */
                 OPEN_FLAGS_NOINHERIT | OPEN_ACCESS_READONLY |
                 OPEN_SHARE_DENYNONE, /* Read access only */
                 0L); /* Extended attributes(ignored) */

    if (rc != NO_ERROR) {
        printf("DosOpen error: return code = %u\n", rc);
        return 1;
    }

    rc = DosQueryFileInfo(hfConfig, /* Handle of file */
                         FIL_STANDARD, /* Request standard (Level 1) info */
                         &fsts3ConfigInfo, /* Buffer for file information */
                         ulBufSize); /* Size of buffer */

    if (rc != NO_ERROR) {
        printf("DosQueryFileInfo error: return code = %u\n", rc);
        return 1;
    }

    rc = DosClose(hfConfig); /* Close the file (check RC in real life) */
    printf("%s --- File size: %u bytes\n", uchFileName, fsts3ConfigInfo.cbFile);
    printf("Last updated: %d/%d/%d; %d:%2.2d\n",
           fsts3ConfigInfo.fdateLastWrite.month, /* Month */
           fsts3ConfigInfo.fdateLastWrite.day, /* Day */
           (fsts3ConfigInfo.fdateLastWrite.year+1980L), /* Years since 1980 */
           fsts3ConfigInfo.ftimeLastWrite.hours, /* Hours */
           fsts3ConfigInfo.ftimeLastWrite.minutes); /* Minutes */

    return NO_ERROR;
}
```

DosQueryFileInfo - Topics

Select an item:

[Syntax](#)

[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DosQueryModFromAddr

DosQueryModFromAddr - Syntax

Returns information about a module address.

```
#include <os2.h>

PVOID      phMod;
PVOID      pObjNum;
ULONG      BuffLen;
PCHAR      pBuff;
PVOID      pOffset;
ULONG      Address;
APIRET     rc;

rc = DosQueryModFromAddr(phMod, pObjNum, BuffLen,
                        pBuff, pOffset, Address);
```

DosQueryModFromAddr Parameter - phMod

phMod (PVOID) - output
The module handle.

DosQueryModFromAddr Parameter - pObjNum

pObjNum (PVOID) - output
The object number.

DosQueryModFromAddr Parameter - BuffLen

BuffLen (ULONG) - in/out
The buffer length.

DosQueryModFromAddr Parameter - pBuff

pBuff (PCHAR) - output
The module name.

DosQueryModFromAddr Parameter - pOffset

pOffset (PVOID) - output
The offset within the object.

DosQueryModFromAddr Parameter - Address

Address (ULONG) - input
The input address.

DosQueryModFromAddr Return Value - rc

rc (APIRET) - returns
Return code descriptions are:

0

NO_ERROR

DosQueryModFromAddr - Parameters

phMod (PVOID) - output
The module handle.

pObjNum (PVOID) - output
The object number.

BuffLen (ULONG) - in/out
The buffer length.

pBuff (PCHAR) - output
The module name.

pOffset (PVOID) - output
The offset within the object.

Address (ULONG) - input
The input address.

rc (APIRET) - returns
Return code descriptions are:

0

NO_ERROR

DosQueryModFromAddr - Remarks

This function takes the passed flat address (supposed to be code, but could also be data or resource) and returns the module handle, module name, and object number. Note that, for the non-shared address (for an .EXE), the handle returned will be for the current context. For debug support, the offset within the object is also returned.

If the address is not found, then the offset in *pOffset* and the object number in *pObjNum* are set to -1, and NO_ERROR is returned.

DosQueryModFromAddr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DosQueryPathInfo

DosQueryPathInfo - Syntax

DosQueryPathInfo gets file information for a file or subdirectory.

```
#define INCL_DOSFILEMGR
#include <os2.h>

PSZ      pszPathName; /* Address of the ASCIIIZ file specification of the file or subdirectory. */
ULONG    ulInfoLevel; /* The level of path information required. */
PVOID     pInfoBuf; /* Address of the storage area containing the requested level of path information. */
ULONG     cbInfoBuf; /* The length, in bytes, of pInfoBuf. */
APIRET    ulrc; /* Return Code. */

ulrc = DosQueryPathInfo(pszPathName, ulInfoLevel,
                        pInfoBuf, cbInfoBuf);
```


DosQueryPathInfo Parameter - pszPathName

pszPathName (PSZ) - input

Address of the ASCIIZ file specification of the file or subdirectory.

Global file-name characters can be used in the name only for level 5 file information.

DosQuerySysInfo is called by an application during initialization to determine the maximum path length allowed by the operating system.

DosQueryPathInfo Parameter - ulInfoLevel

ulInfoLevel (ULONG) - input

The level of path information required.

A value of 1, 2, 3, 4, or 5 can be specified, as follows:

1	FIL_STANDARD Level 1 file information
2	FIL_QUERYEASIZE Level 2 file information
3	FIL_QUERYEASFROMLIST Level 3 file information
4	FIL_QUERYEASALL Level-4 file information
5	FIL_QUERYFULLNAME Level 5 file information

The structures described in *pInfoBuf* indicate the information returned for each of these levels.

DosQueryPathInfo Parameter - pInfoBuf

pInfoBuf (PVOID) - output

Address of the storage area containing the requested level of path information.

Path information, where applicable, is based on the most recent DosClose, DosResetBuffer, DosSetFileInfo, or DosSetPathInfo.

Level 1 File Information (*ulInfoLevel* == FIL_STANDARD)

pInfoBuf contains the [FILESTATUS3](#) data structure, in which path information is returned.

Level 2 File Information (*ulInfoLevel* == FIL_QUERYEASIZE)

pInfoBuf contains the [FILESTATUS4](#) data structure. This is similar to the Level 1 structure, with the addition of the *cbList* field after the *attrFile* field.

The *cbList* field is an unsigned ULONG. On output, this field contains the size, in bytes, of the file's entire extended attribute (EA) set on disk. You can use this value to calculate the size of the buffer required to hold the EA information returned when a value of 3 is specified for *ulInfoLevel*. The buffer size is less than or equal to twice the size of the file's entire EA set on disk.

Level 3 File Information (*ulInfoLevel* == FIL_QUERYEASFROMLIST)

This is a subset of the EA information of the file.

Input	<i>ulInfoLevel</i> contains an EAOP2 data structure. <i>fpGEA2List</i> points to a GEA2 that defines the attribute names whose values are returned. The GEA2 data structures must be aligned on a doubleword boundary. Each <i>oNextEntryOffset</i> field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry in the GEA2 list. The <i>oNextEntryOffset</i> field in the last entry of the GEA2 list must be zero. <i>fpFEA2List</i> points to a data area where the relevant FEA2 list is returned. The length field of this FEA2 list is valid, giving the size of the FEA2 list buffer. <i>oError</i> is ignored.
Output	<i>pInfoBuf</i> is unchanged. If an error occurs, <i>oError</i> points to the GEA2 entry that caused the error. The buffer pointed to by <i>fpFEA2List</i> is filled in with the returned information. If the buffer that <i>fpFEA2List</i> points to is not large enough to hold the returned information (the return code is <code>ERROR_BUFFER_OVERFLOW</code>), <i>cbList</i> is still valid, assuming there is at least enough space for it. Its value is the size, in bytes, of the file's entire EA set on disk, even though only a subset of attributes was requested. The size of the buffer required to hold the EA information is less than or equal to twice the size of the file's entire EA set on disk.
Level 5 File Information (<i>ulInfoLevel</i> == <code>FIL_QUERYFULLNAME</code>)	
Level 5 returns the fully qualified ASCIIZ name of <i>pszPathName</i> in <i>pInfoBuf</i> . <i>pszPathName</i> may contain global file-name characters.	

DosQueryPathInfo Parameter - cbInfoBuf

cbInfoBuf (ULONG) - input
The length, in bytes, of *pInfoBuf*.

DosQueryPathInfo Return Value - ulrc

ulrc (APIRET) - returns
Return Code.

DosQueryPathInfo returns one of the following values:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
32	ERROR_SHARING_VIOLATION
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
206	ERROR_FILENAME_EXCED_RANGE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

DosQueryPathInfo - Parameters

pszPathName (PSZ) - input
Address of the ASCIIZ file specification of the file or subdirectory.

Global file-name characters can be used in the name only for level 5 file information.

DosQuerySysInfo is called by an application during initialization to determine the maximum path length allowed by the operating system.

ulInfoLevel (ULONG) - input

The level of path information required.

A value of 1, 2, 3, 4, or 5 can be specified, as follows:

1	FIL_STANDARD Level 1 file information
2	FIL_QUERYEASIZE Level 2 file information
3	FIL_QUERYEASFROMLIST Level 3 file information
4	FIL_QUERYEASALL Level-4 file information
5	FIL_QUERYFULLNAME Level 5 file information

The structures described in *pInfoBuf* indicate the information returned for each of these levels.

pInfoBuf (PVOID) - output

Address of the storage area containing the requested level of path information.

Path information, where applicable, is based on the most recent `DosClose`, `DosResetBuffer`, `DosSetFileInfo`, or `DosSetPathInfo`.

Level 1 File Information (*ulInfoLevel* == FIL_STANDARD)

pInfoBuf contains the [FILESTATUS3](#) data structure, in which path information is returned.

Level 2 File Information (*ulInfoLevel* == FIL_QUERYEASIZE)

pInfoBuf contains the [FILESTATUS4](#) data structure. This is similar to the Level 1 structure, with the addition of the *cbList* field after the *attrFile* field.

The *cbList* field is an unsigned ULONG. On output, this field contains the size, in bytes, of the file's entire extended attribute (EA) set on disk. You can use this value to calculate the size of the buffer required to hold the EA information returned when a value of 3 is specified for *ulInfoLevel*. The buffer size is less than or equal to twice the size of the file's entire EA set on disk.

Level 3 File Information (*ulInfoLevel* == FIL_QUERYEASFROMLIST)

This is a subset of the EA information of the file.

Input *ulInfoLevel* contains an [EAOP2](#) data structure. *fpGEA2List* points to a [GEA2](#) that defines the attribute names whose values are returned. The [GEA2](#) data structures must be aligned on a doubleword boundary. Each *oNextEntryOffset* field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry in the [GEA2](#) list. The *oNextEntryOffset* field in the last entry of the [GEA2](#) list must be zero. *fpFEA2List* points to a data area where the relevant [FEA2](#) list is returned. The length field of this [FEA2](#) list is valid, giving the size of the [FEA2](#) list buffer. *oError* is ignored.

Output *pInfoBuf* is unchanged. If an error occurs, *oError* points to the [GEA2](#) entry that caused the error. The buffer pointed to by *fpFEA2List* is filled in with the returned information. If the buffer that *fpFEA2List* points to is not large enough to hold the returned information (the return code is `ERROR_BUFFER_OVERFLOW`), *cbList* is still valid, assuming there is at least enough space for it. Its value is the size, in bytes, of the file's entire EA set on disk, even though only a subset of attributes was requested. The size of the buffer required to hold the EA information is less than or equal to twice the size of the file's entire EA set on disk.

Level 5 File Information (*ulInfoLevel* == FIL_QUERYFULLNAME)

Level 5 returns the fully qualified ASCIIZ name of *pszPathName* in *pInfoBuf*. *pszPathName* may contain global file-name characters.

cbInfoBuf (ULONG) - input

The length, in bytes, of *pInfoBuf*.

ulrc (APIRET) - returns

Return Code.

`DosQueryPathInfo` returns one of the following values:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
32	ERROR_SHARING_VIOLATION

111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
206	ERROR_FILENAME_EXCED_RANGE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

DosQueryPathInfo - Remarks

For DosQueryPathInfo to return information contained in any of the file information levels, the file object must be opened for read access, with a deny-write sharing mode specified for access by other processes. Thus, if the file object is already accessed by another process that holds conflicting sharing and access rights, a call to DosQueryPathInfo fails.

DosQueryPathInfo - Related Functions

- DosClose
- DosCreateDir
- DosEnumAttribute
- DosOpen
- [DosQueryFileInfo](#)
- DosResetBuffer
- DosSetFileInfo
- DosSetPathInfo

DosQueryPathInfo - Example Code

The first example obtains information about the file "STARTUP.CMD." The second example obtains information about the directory "SYSTEM."

```
#define INCL_DOSFILEMGR    /* File Manager values */
#define INCL_DOSERRORS    /* DOS error values */
#include <os2.h>
#include <stdio.h>

int main(VOID) {
    UCHAR      uchFileName[80] = "C:\\\\STARTUP.CMD"; /* File to manipulate */
    FILESTATUS3 fsts3ConfigInfo = {{0}};           /* Buffer for file information */
    ULONG      ulBufSize      = sizeof(FILESTATUS3); /* Size of above buffer */
    APIRET     rc              = NO_ERROR;          /* Return code */

    rc = DosQueryPathInfo(uchFileName, /* Path and name of file */
                          FIL_STANDARD, /* Request standard (Level 1) info */
                          &fsts3ConfigInfo, /* Buffer for file information */
                          ulBufSize); /* Size of buffer */

    if (rc != NO_ERROR) {
        printf("DosQueryPathInfo error: return code = %u\n", rc);
        return 1;
    }

    printf("%s --- File size: %u bytes\n", uchFileName, fsts3ConfigInfo.cbFile);
    printf("Last updated: %d/%d/%d; %d:%2.2d\n",
        fsts3ConfigInfo.fdateLastWrite.month, /* Month */
        fsts3ConfigInfo.fdateLastWrite.day, /* Day */
        (fsts3ConfigInfo.fdateLastWrite.year+1980L), /* Years since 1980 */
        fsts3ConfigInfo.ftimeLastWrite.hours, /* Hours */
        fsts3ConfigInfo.ftimeLastWrite.minutes); /* Minutes */

    return NO_ERROR;
}
```

```

#define INCL_DOSFILEMGR    /* File Manager values */
#define INCL_DOSERRORS    /* DOS error values */
#include <os2.h>
#include <stdio.h>

int main(VOID) {
    UCHAR        uchPathName[255] = "C:\\OS2\\SYSTEM"; /* Path of interest */
    FILESTATUS3   fsts3ConfigInfo = {{0}}; /* Buffer for path information */
    ULONG         ulBufSize       = sizeof(FILESTATUS3); /* Size of above buffer */
    APIRET        rc              = NO_ERROR; /* Return code */

    rc = DosQueryPathInfo(uchPathName, /* Name of path */
                          FIL_STANDARD, /* Request standard (Level 1) info */
                          &fsts3ConfigInfo, /* Buffer for information */
                          ulBufSize); /* Size of buffer */

    if (rc != NO_ERROR) {
        printf("DosQueryPathInfo error: return code = %u\n", rc);
        return 1;
    }

    printf("Information for subdirectory: %s:\n", uchPathName);
    printf("Last updated: %d/%d/%d; %d:%2.2d\n",
           fsts3ConfigInfo.fdateLastWrite.month, /* Month */
           fsts3ConfigInfo.fdateLastWrite.day, /* Day */
           (fsts3ConfigInfo.fdateLastWrite.year+1980L), /* Years since 1980 */
           fsts3ConfigInfo.ftimeLastWrite.hours, /* Hours */
           fsts3ConfigInfo.ftimeLastWrite.minutes); /* Minutes */

    return NO_ERROR;
}

```

DosQueryPathInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DosQueryProcessInfo

DosQueryProcessInfo - Syntax

DosQueryProcessInfo will return information about processes and tasks running in the system.

```

#define INCL_WPOS
#include <os2.h>

ULONG      InfoType;
ULONG      ID;
ULONG      InfoLevel;
PVOID      InfoBuffer;
PULONG     InfoBufferLength;
PULONG     InfoEntryCount;

```

```
APIRET      ulrc;                /* Return code. */

ulrc = DosQueryProcessInfo(InfoType, ID, InfoLevel,
                          InfoBuffer, InfoBufferLength, InfoEntryCount);
```

DosQueryProcessInfo Parameter - InfoType

InfoType (ULONG) - input

Indicates the kind of processes for which information is requested.

Must be one of the following values:

QPI_PID	Specified process ID/task ID
QPI_OS2_USER	OS/2 processes of this user
QPI_SYSTEM_ALL	OS/2 processes of all the users
QPI_SYSTEM_USER	OS/2 and other tasks for this user
QPI_OS2_ALL	OS/2 and other tasks for all users.

DosQueryProcessInfo Parameter - ID

ID (ULONG) - input

The process ID or task ID for which information is requested. If this is zero, then information for the current process is returned. This can also be the task ID of a non-OS/2 task. This is only used if QPI_PID(InfoType 0) is specified in the InfoType parameter.

DosQueryProcessInfo Parameter - InfoLevel

InfoLevel (ULONG) - input

The level of information to return.

Must be one of the following values:

QPI_BASIC	Basic information
QPI_EXTENDED	Extended and library information. Extended information causes the system to perform internal queries and is slow.

DosQueryProcessInfo Parameter - InfoBuffer

InfoBuffer (PVOID) - output

Location to return the output information. The buffer must be 4-byte (32-bit) aligned.

DosQueryProcessInfo Parameter - InfoBufferLength

InfoBufferLength (PULONG) - in/out

Input - Size of the InfoBuffer.
Output - The size of the buffer returned.

If the caller's buffer is too small, then this will be set to the required size.

DosQueryProcessInfo Parameter - InfoEntryCount

InfoEntryCount (PULONG) - output
The count of entries returned.

DosQueryProcessInfo Return Value - ulrc

ulrc (APIRET) - returns
Return code.

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
303	ERROR_INVALID_PROCID
309	ERROR_INVALID_THREADID

DosQueryProcessInfo - Parameters

InfoType (ULONG) - input
Indicates the kind of processes for which information is requested.

Must be one of the following values:

QPI_PID	Specified process ID/task ID
QPI_OS2_USER	OS/2 processes of this user
QPI_SYSTEM_ALL	OS/2 processes of all the users
QPI_SYSTEM_USER	OS/2 and other tasks for this user
QPI_OS2_ALL	OS/2 and other tasks for all users.

ID (ULONG) - input

The process ID or task ID for which information is requested. If this is zero, then information for the current process is returned. This can also be the task ID of a non-OS/2 task. This is only used if QPI_PID(InfoType 0) is specified in the InfoType parameter.

InfoLevel (ULONG) - input

The level of information to return.

Must be one of the following values:

QPI_BASIC

Basic information

QPI_EXTENDED

Extended and library information. Extended information causes the system to perform internal queries and is slow.

InfoBuffer (PVOID) - output

Location to return the output information. The buffer must be 4-byte (32-bit) aligned.

InfoBufferLength (PULONG) - in/out

Input - Size of the InfoBuffer.

Output - The size of the buffer returned.

If the caller's buffer is too small, then this will be set to the required size.

InfoEntryCount (PULONG) - output

The count of entries returned.

ulrc (APIRET) - returns

Return code.

0

NO_ERROR

8

ERROR_NOT_ENOUGH_MEMORY

87

ERROR_INVALID_PARAMETER

111

ERROR_BUFFER_OVERFLOW

303

ERROR_INVALID_PROCID

309

ERROR_INVALID_THREADID

DosQueryProcessInfo - Remarks

The information returned depends on the InfoLevel requested. The first entry is a length field which gives the offset in bytes to the next entry. Variable-length information is placed at the end of the fixed structure.

All entries returned will 4-byte aligned and the buffer must be 4-byte aligned.

The QPROCINFO1 structure will be returned for QPI_BASIC (InfoLevel = 1) queries. QPROCINFO1 will be repeated InfoEntryCount times in the InfoBuffer. Session id, session type, priority and hmodule information will be returned for OS/2 processes only.

The QPROCINFO2 structure will be returned for QPI_EXTENDED (InfoLevel = 2) queries. QPROCINFO2 will be repeated InfoEntryCount times in the InfoBuffer. Session id, session type and hmodule information will be returned for OS/2 processes only.

DosQueryProcessInfo - Example Code

The following example will query the system for current OS/2 process basic information. Since the query is for the current process, upon return no_of_entries will be set to 1.

```
#define INCL_WPOS
```



```

APIRET rc;
ULONG buffer_length, no_of_entries;
char_buffer[16000];

buffer_length = 16000;
rc = DosQueryProcInfo( QPI_PID, 0, QPI_BASIC, &char_buffer,
                      &buffer_length, &no_of_entries);

if (rc!=0) /* If problem print the reason code */
{
    printf("DosQueryProcInfo Error: return code = %ld", rc);
}

return;

```

DosQueryProcessInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DosQueryThreadInfo

DosQueryThreadInfo - Syntax

DosQueryThreadInfo will return information about threads for the specified process or task.

```

#define INCL_WPOS
#include <os2.h>

ULONG ThreadID;
ULONG ID;
ULONG InfoLevel;
PVOID InfoBuffer;
PULONG InfoBufferLength;
PULONG InfoEntryCount;
APIRET ulrc; /* Return code. */

ulrc = DosQueryThreadInfo(ThreadID, ID, InfoLevel,
                          InfoBuffer, InfoBufferLength, InfoEntryCount);

```

DosQueryThreadInfo Parameter - ThreadID

ThreadID (ULONG) - input

The thread ID for which information is requested. If this is zero, then information for all threads in the process will be returned.

DosQueryThreadInfo Parameter - ID

ID (ULONG) - input

The process ID (or task ID for a non-OS/2 task) for which information is requested. If this is set to zero, then information for the current process will be returned.

DosQueryThreadInfo Parameter - InfoLevel

InfoLevel (ULONG) - input

The level of information to return.

Must be one of the following values:

QTI_BASIC

Basic information

QTI_EXTENDED

Extended information. Extended information causes the system to perform internal queries and is slow.

DosQueryThreadInfo Parameter - InfoBuffer

InfoBuffer (PVOID) - output

Location to return the output information. The buffer must be 4-byte (32-bit) aligned.

DosQueryThreadInfo Parameter - InfoBufferLength

InfoBufferLength (PULONG) - in/out

Input - Size of the InfoBuffer.

Output - The size of the buffer returned.

If the caller's buffer is too small, then this will be set to the required size.

DosQueryThreadInfo Parameter - InfoEntryCount

InfoEntryCount (PULONG) - output

The count of entries returned.

DosQueryThreadInfo Return Value - ulrc

ulrc (APIRET) - returns
Return code.

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
303	ERROR_INVALID_PROCID
309	ERROR_INVALID_THREADID

DosQueryThreadInfo - Parameters

ThreadID (ULONG) - input

The thread ID for which information is requested. If this is zero, then information for all threads in the process will be returned.

ID (ULONG) - input

The process ID (or task ID for a non-OS/2 task) for which information is requested. If this is set to zero, then information for the current process will be returned.

InfoLevel (ULONG) - input

The level of information to return.

Must be one of the following values:

QTI_BASIC	Basic information
QTI_EXTENDED	Extended information. Extended information causes the system to perform internal queries and is slow.

InfoBuffer (PVOID) - output

Location to return the output information. The buffer must be 4-byte (32-bit) aligned.

InfoBufferLength (PULONG) - in/out

Input - Size of the InfoBuffer.
Output - The size of the buffer returned.

If the caller's buffer is too small, then this will be set to the required size.

InfoEntryCount (PULONG) - output

The count of entries returned.

ulrc (APIRET) - returns
Return code.

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
303	

DosQueryThreadInfo - Remarks

The information returned depends on the InfoLevel requested. The first entry is a length field which gives the offset in bytes to the next entry. Variable-length information is placed at the end of the fixed structure.

All entries returned will 4-byte aligned and the buffer must be 4-byte aligned.

The QTHREADINFO1 structure will be returned for QTI_BASIC (InfoLevel = 1) queries. QTHREADINFO1 will be repeated InfoEntryCount times in the InfoBuffer.

The QTHREADINFO2 structure will be returned for QTI_EXTENDED (InfoLevel = 2) queries. QTHREADINFO2 will be repeated InfoEntryCount times in the InfoBuffer.

DosQueryThreadInfo - Example Code

The following example will query the system for thread information in the current OS/2 process.

```
#define          INCL_WPOS

APIRET rc;
ULONG  buffer_length, no_of_entries;
char_buffer[16000];

buffer_length = 16000;
rc = DosQueryThreadInfo( 0, 0, QTI_BASIC, &char_buffer,
                        &buffer_length, &no_of_entries);

if (rc!=0) /* If problem print the reason code */
{
    printf("DosQueryThreadInfo Error: return code = %ld", rc);
}

return;
```

DosQueryThreadInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DosQueryUconvObject

DosQueryUconvObject - Syntax

Retrieves the conversion object for the current process.

```
<os2.h>
<uconv.h >

UconvObject      *uconv_object;
APIRET           return_value; /* Return code. */

return_value = DosQueryUconvObject(uconv_object);
```

DosQueryUconvObject Parameter - uconv_object

uconv_object (UconvObject *) - in/out
A pointer to a conversion object upon successful completion.

DosQueryUconvObject Return Value - return value

return_value (APIRET) - returns
Return code.

DosQueryUconvObject returns one of the following values:

0	Indicates success.
103	A conversion object does not exist.

DosQueryUconvObject - Parameters

uconv_object (UconvObject *) - in/out
A pointer to a conversion object upon successful completion.

return_value (APIRET) - returns
Return code.

DosQueryUconvObject returns one of the following values:

0	Indicates success.
103	A conversion object does not exist.

DosQueryUconvObject - Remarks

DosQueryUconvObject retrieves the conversion object of the current process. This conversion object can be used in subsequent calls to UniUconvFromUcs and UniUconvToUcs. The conversion object will convert between UCS-2 (Unicode) and the current OS/2 code page.

The returned object should not be given to the UniFreeUconvObject function when it is no longer needed. **Do not** discard the object when it is no longer needed. This object is shared.

DosQueryUconvObject is supplied to speed up the conversion process for those implementations converting between the current code page and Unicode.

DosQueryUconvObject - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

DosQueryUnicodeCpString

DosQueryUnicodeCpString - Syntax

Queres the code page of the current OS/2 process.

```
#define INCL_<os2.h>
<unidef.h>

UniChar      *code_set;
int           max;
APIRET       return value;

return value = DosQueryUnicodeCpString(code_set,
                                       max);
```

DosQueryUnicodeCpString Parameter - code_set

code_set (UniChar *) - in/out

A pointer to a buffer to hold the returned code-set name.

DosQueryUnicodeCpString Parameter - max

max (int) - in/out
The length of *code_set* (number of UniChar code elements, not bytes, in the *code_set* buffer).

DosQueryUnicodeCpString Return Value - return value

return value (APIRET) - returns
Return codes.

0	Indicates success.
UCONV_E2BIG	Output buffer <i>code_set</i> form=textonly. is not large enough.

DosQueryUnicodeCpString - Parameters

code_set (UniChar *) - in/out
A pointer to a buffer to hold the returned code-set name.

max (int) - in/out
The length of *code_set* (number of UniChar code elements, not bytes, in the *code_set* buffer).

return value (APIRET) - returns
Return codes.

0	Indicates success.
UCONV_E2BIG	Output buffer <i>code_set</i> form=textonly. is not large enough.

DosQueryUnicodeCpString - Remarks

DosQueryUnicodeCpString queries the code page of the current OS/2 process. A string of type (UniChar form=textonly. *) representing the code page is returned in *code_set* , which can be used in a subsequent call to UniCreateUconvObject to create a conversion object.

DosQueryUnicodeCpString - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DosReplaceModule

DosReplaceModule - Syntax

DosReplaceModule replaces an active DLL or EXE by caching the contents of the old module and then copying a new file over the old module's file.

```
#include <os2.h>

ULONG      pszOldModule;
ULONG      pszNewModule;
ULONG      pszBackupModule;
APIRET     rc;

rc = DosReplaceModule(pszOldModule, pszNewModule,
                      pszBackupModule);
```

DosReplaceModule Parameter - pszOldModule

pszOldModule (ULONG) - input
Name of the module to be replaced.

DosReplaceModule Parameter - pszNewModule

pszNewModule (ULONG) - in/out
Name of the module to replace the old module.

DosReplaceModule Parameter - pszBackupModule

pszBackupModule (ULONG) - input
Backup module name.

DosReplaceModule Return Value - rc

rc (APIRET) - returns
Return code descriptions are:

0
Successful.

DosReplaceModule - Parameters

pszOldModule (ULONG) - input
Name of the module to be replaced.

pszNewModule (ULONG) - in/out
Name of the module to replace the old module.

pszBackupModule (ULONG) - input
Backup module name.

rc (APIRET) - returns
Return code descriptions are:

0
Successful.

DosReplaceModule - Remarks

Just in case something goes wrong, a backup copy of the old module's file is made before the new module is copied over it. If an error occurs, the backup copy is used to replace the bad new copy of the module.

If *pszNewModule* or *pszBackupModule* is NULL, no action is taken. With both of these NULL, the result is just to cache and close the file.

DosReplaceModule - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DosSelectSession

DosSelectSession - Syntax

DosSelectSession allows one session to switch another session to the foreground.

```
#define INCL_dossesmgr  
#include <os2.h>
```

```
ULONG      idSession; /* The identifier of the session to be switched to the foreground. */  
APIRET     ulrc;      /* Return Code. */
```

```
ulrc = DosSelectSession(idSession);
```

DosSelectSession Parameter - idSession

idSession (ULONG) - input

The identifier of the session to be switched to the foreground.

The value specified must have been returned on a previous call to DosStartSession, except that a value of zero indicates switching the caller's session to the foreground.

DosSelectSession Return Value - ulrc

ulrc (APIRET) - returns

Return Code.

DosSelectSession returns one of the following values:

0	NO_ERROR
224	ERROR_SMG_NO_TARGET_WINDOW
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
459	ERROR_SMG_BAD_RESERVE
463	ERROR_SMG_RETRY_SUB_ALLOC

DosSelectSession - Parameters

idSession (ULONG) - input

The identifier of the session to be switched to the foreground.

The value specified must have been returned on a previous call to DosStartSession, except that a value of zero indicates switching the caller's session to the foreground.

ulrc (APIRET) - returns

Return Code.

DosSelectSession returns one of the following values:

0	NO_ERROR
224	ERROR_SMG_NO_TARGET_WINDOW
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
459	ERROR_SMG_BAD_RESERVE
463	ERROR_SMG_RETRY_SUB_ALLOC

DosSelectSession - Remarks

DosSelectSession allows a session to switch another session to the foreground. The session specified will not be brought to the foreground

unless the calling session is currently executing in the foreground.

The foreground session for windowed applications is the session of the application that owns the window focus.

When DosSelectSession is issued, the session specified will not be brought to the foreground unless the calling session or one of its descendant sessions is currently executing in the foreground.

Return code ERROR_SMG_NO_TARGET_WINDOW is a warning that the session might not be brought to the foreground. If the selected session is a Presentation Manager (PM) application, its window must be created with the FCF_TASKLIST flag bit set on. If the window is created with this bit set off, its session cannot be selected using DosSelectSession, and ERROR_SMG_NO_TARGET_WINDOW is returned.

If you issue DosSelectSession before creating the PM window of the selected session, ERROR_SMG_NO_TARGET_WINDOW is returned. However, if the PM window of the selected session is subsequently created with the FCF_TASKLIST flag bit set on, the window is brought to the foreground if the issuer of DosSelectSession still owns the foreground focus.

If a session still exists but its window has been destroyed, and you issue DosSelectSession for that session, ERROR_SMG_NO_TARGET_WINDOW is returned.

DosSelectSession - Related Functions

- [DosSetSession](#)
 - [DosStartSession](#)
 - [DosStopSession](#)
-

DosSelectSession - Example Code

This example shows how to bring a child process to the foreground.

```
#define INCL_DOSPROCESS    /* DOS process values - needed for DosSleep */
#define INCL_DOSSESMGR
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>

int main(VOID) {
    STARTDATA SData      = {0};
    PSZ        PgmTitle   = "PEEK-A-BOO!   I see you!", /* Title      */
    PgmName     = "CMD.EXE"; /* This starts an OS/2 session */
    APIRET      rc         = NO_ERROR; /* Return code */
    PID         pid        = 0; /* PID returned */
    ULONG       ulSessID   = 0; /* Session ID returned */
    UCHAR       achObjBuf[100] = {0};

    SData.Length = sizeof(STARTDATA);
    SData.Related = SSF_RELATED_CHILD; /* start a dependent session */
    SData.FgBg    = SSF_FGBG_BACK; /* start session in background */
    SData.TraceOpt = SSF_TRACEOPT_NONE; /* No trace */
    /* Start an OS/2 session using "CMD.EXE /K" */
    SData.PgmTitle = PgmTitle;
    SData.PgmName = PgmName;
    SData.PgmInputs = "/K"; /* Keep session up */
    SData.TermQ = 0; /* No termination queue */
    SData.Environment = 0; /* No environment string */
    SData.InheritOpt = SSF_INHERTOPT_SHELL; /* Inherit shell's environ. */
    SData.SessionType = SSF_TYPE_WINDOWABLEVIO; /* Windowed VIO session */
    SData.IconFile = 0; /* No icon association */
    SData.PgmHandle = 0;
    SData.PgmControl = SSF_CONTROL_VISIBLE | SSF_CONTROL_MAXIMIZE;
    SData.InitXPos = 30; /* Initial window coordinates */
    SData.InitYPos = 40;
    SData.InitXSize = 200; /* Initial window size */
    SData.InitYSize = 140;
    SData.Reserved = 0;
    SData.ObjectBuffer = achObjBuf; /* Contains info if DosExecPgm fails */
    SData.ObjectBuffLen = (ULONG) sizeof(achObjBuf);

    rc = DosStartSession(&SData, &ulSessID, &pid); /* Start the session */
}
```

```

if (rc != NO_ERROR) {
    printf ("DosStartSession error : return code = %u\n", rc);
    return 1;
}

printf("Child session will appear in the foreground in a moment.\n");
rc = DosSleep(2500L);

rc = DosSelectSession(ulSessID);
if (rc != NO_ERROR) {
    printf ("DosSelectSession error : return code = %u\n", rc);
    return 1;
}
rc = DosSleep(5000L);
return NO_ERROR;
}

```

DosSelectSession - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DosSetProcessCp

DosSetProcessCp - Syntax

This function sets the country code of the current OS/2 process.

```

#include <os2.h>

PULONG    CountryCode;
APIRET    rc;

rc = DosSetProcessCp(CountryCode);

```

DosSetProcessCp Parameter - CountryCode

CountryCode (PULONG) - output

The returned country code of the current process.

DosSetProcessCp Return Value - rc

rc (APIRET) - returns

Return code descriptions are:

0

NO_ERROR

DosSetProcessCp - Parameters

CountryCode (PULONG) - output

The returned country code of the current process.

rc (APIRET) - returns

Return code descriptions are:

0

NO_ERROR

DosSetProcessCp - Remarks

DosSetProcessCp sets the process country code of the calling process. The country code of a process is inherited by a newly created child process.

DosSetProcessCp does not affect the code-page setting of the current process.

DosSetProcessCp - Example Code

```
#define INCL_DOSNLS    /* National Language Support values */
#include <os2.h>
#include <stdio.h>

ULONG CountryCode = 001;
APIRET rc;          /* Return code */

rc = DosSetProcessCp( CountryCode );

if (rc != 0)
{
    printf("DosSetProcessCp error: return code = %ld",rc);
    return;
}
```

DosSetProcessCp - Topics

Select an item:

DosSetDosProperty

DosSetDosProperty - Syntax

DosSetDosProperty allows an OS/2 session to set a DOS property.

```
#include <os2.h>

SGID      SGID;
PSZ       PszName;
ULONG     cb;
PSZ       pch;
APIRET    rc;

rc = DosSetDosProperty(SGID, PszName, cb,
                       pch);
```

DosSetDosProperty Parameter - SGID

SGID ([SGID](#)) - input
Session identification.

DosSetDosProperty Parameter - PszName

PszName ([PSZ](#)) - input
Property name.

DosSetDosProperty Parameter - cb

cb ([ULONG](#)) - output
Buffer length.

DosSetDosProperty Parameter - pch

pch (PSZ) - input
Property value.

DosSetDosProperty Return Value - rc

rc (APIRET) - returns
Return codes.

0
NO_ERROR

DosSetDosProperty - Parameters

SGID ([SGID](#)) - input
Session identification.

PszName (PSZ) - input
Property name.

cb (ULONG) - output
Buffer length.

pch (PSZ) - input
Property value.

rc (APIRET) - returns
Return codes.

0
NO_ERROR

DosSetDosProperty - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

DosSetSession

DosSetSession - Syntax

DosSetSession sets the status of another session.

```
#define INCL_dossemgr
#include <os2.h>

ULONG      idSession; /* The identifier of the target session. */
PSTATUSDATA psd;      /* A pointer to the STATUSDATA which contains the session status data. */
APIRET     ulrc;       /* Return code. */

ulrc = DosSetSession(idSession, psd);
```

DosSetSession Parameter - idSession

idSession (ULONG) - input

The identifier of the target session.

The value specified must have been returned on a previous call to DosStartSession.

DosSetSession Parameter - psd

psd (PSTATUSDATA) - input

A pointer to the STATUSDATA which contains the session status data.

DosSetSession Return Value - ulrc

ulrc (APIRET) - returns

Return code.

DosSetSession returns one of the following values:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
455	ERROR_SMG_INVALID_BOND_OPTION
456	ERROR_SMG_INVALID_SELECT_OPT
461	ERROR_SMG_INVALID_DATA_LENGTH
463	ERROR_SMG_RETRY_SUB_ALLOC

DosSetSession - Parameters

idSession (ULONG) - input

The identifier of the target session.

The value specified must have been returned on a previous call to DosStartSession.

psd (PSTATUSDATA) - input

A pointer to the [STATUSDATA](#) which contains the session status data.

ulrc (APIRET) - returns

Return code.

DosSetSession returns one of the following values:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
455	ERROR_SMG_INVALID_BOND_OPTION
456	ERROR_SMG_INVALID_SELECT_OPT
461	ERROR_SMG_INVALID_DATA_LENGTH
463	ERROR_SMG_RETRY_SUB_ALLOC

DosSetSession - Remarks

DosSetSession sets or resets one or both of the following parameters related to another session:

1. Selectable or nonselectable. This parameter allows the calling session to set another session as selectable or nonselectable from the Shell switch list.
2. Bond/no bond. This parameter allows the calling session to bond another session to itself. This means that if the operator subsequently selects the calling session from the Shell menu (or double-clicks to the calling session), then the bonded session will be brought to the foreground.

The parameters only affect user selections from the Shell switch list or Shell selections during system hot key processing. They do not affect selections made by the calling session. Thus, when the calling session selects its own session, its own session is brought to the foreground, even if a bond is in effect. Likewise, when a session selects a nonselectable session, the nonselectable session is brought to the foreground.

The above parameters may be set individually. Either can be changed without affecting the current setting of the other.

A bond established between two sessions can be broken by reissuing DosSetSession and specifying either:

- *BondInd* = 2 to break the bond, or
- *BondInd* = 1 to establish a bond with a different session. In this case, the bond with the previous session is broken.

If a bond is established between session A and session B, and if another bond is established between session B and session C, then if the operator selects session A, session C is brought to the foreground. However, if session A selects itself, session A is brought to the foreground. If session A selects session B, session C is brought to the foreground. In the latter case, the bond between B and C is honored.

Assume that a bond is established between session A and session B, and assume that session B is nonselectable. The operator will not be able to select session B directly. However, if the operator selects session A, session B will be brought to the foreground.

The calling session may be running in either the foreground or the background when DosSetSession is issued.

DosSetSession - Related Functions

- [DosSelectSession](#)
- [DosStartSession](#)
- [DosStopSession](#)

DosSetSession - Example Code

This example removes a child session from a window list.

```
#define INCL_DOSPROCESS    /* DOS process values - needed for DosSleep */
#define INCL_DOSSESMGR
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>

int main(VOID) {
    STARTDATA SData      = {0};
    PSZ        PgmTitle   = "Not in the Window List",    /* Title */
    PgmName     = "CMD.EXE"; /* This starts an OS/2 session */
    APIRET      rc         = NO_ERROR; /* Return code */
    PID         pid        = 0; /* PID returned */
    ULONG       ulSessID   = 0; /* Session ID returned */
    UCHAR       achObjBuf[100] = {0}; /* Error info if start fails */
    STATUSDATA  ChildStatus = {0}; /* Child status data */

    SData.Length = sizeof(STARTDATA);
    SData.Related = SSF_RELATED_CHILD; /* A dependent session */
    SData.FgBg    = SSF_FGBG_FORE; /* start session in foreground */
    SData.TraceOpt = SSF_TRACEOPT_NONE; /* No trace */
    /* Start an OS/2 session using "CMD.EXE /K" */
    SData.PgmTitle = PgmTitle;
    SData.PgmName = PgmName;
    SData.PgmInputs = "/K"; /* Keep session up */
    SData.TermQ = 0; /* No termination queue */
    SData.Environment = 0; /* No environment string */
    SData.InheritOpt = SSF_INHERTOPT_SHELL; /* Inherit shell's environ. */
    SData.SessionType = SSF_TYPE_WINDOWABLEVIO; /* Windowed VIO session */
    SData.IconFile = 0; /* No icon association */
    SData.PgmHandle = 0;
    SData.PgmControl = SSF_CONTROL_VISIBLE | SSF_CONTROL_MAXIMIZE;
    SData.InitXPos = 30; /* Initial window coordinates */
    SData.InitYPos = 40;
    SData.InitXSize = 200; /* Initial window size */
    SData.InitYSize = 140;
    SData.Reserved = 0;
    SData.ObjectBuffer = achObjBuf; /* Contains info if DosExecPgm fails */
    SData.ObjectBuffLen = (ULONG) sizeof(achObjBuf);
    rc = DosStartSession(&SData, &ulSessID, &pid); /* Start the session */
    if (rc != NO_ERROR) {
        printf ("DosStartSession error : return code = %u\n", rc);
        return 1;
    }
    printf("Removing child process from the Window List... \n");

    ChildStatus.Length = sizeof(STATUSDATA);
    ChildStatus.SelectInd = SET_SESSION_NON_SELECTABLE;
    ChildStatus.BondInd = SET_SESSION_UNCHANGED;

    rc = DosSetSession(ulSessID, &ChildStatus);
    if (rc != NO_ERROR) {
        printf ("DosSetSession error : return code = %u\n", rc);
        return 1;
    }

    printf("\nPress Ctrl-Esc The child is no longer listed.\n");
    printf("\nProgram will terminate in 10 seconds...\n");
    rc = DosSleep(10000L); /* wait 10 seconds */

    return NO_ERROR;
}
```

DosSetSession - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)

DosShutdown

DosShutdown - Syntax

DosShutdown locks out changes to all file systems, and writes system buffers to the disk in preparation for turning off power to the system.

```
#define INCL_DOSFILEMGR
#include <os2.h>

ULONG      ShutDownValue; /* Flag indicating the shutdown value. */
APIRET     ulrc;          /* Return code. */

ulrc = DosShutdown(ShutDownValue);
```

DosShutdown Parameter - ShutDownValue

ShutDownValue (ULONG) - input
Flag indicating the shutdown value.

ShutDownValue has the following states:

<u>Bit</u>	<u>Description</u>
31-16	x00 Reserved.
15-8	Shutdown Type: x00 Return to caller x01 Halt x02 Reboot
7-4	x00 Reserved
3-0	Actions: x00 Flush file system buffers and caches. x01 Flush file system buffers/caches. x02 System dump is requested. The file system will no longer accept write requests.

If a system dump is requested, no shutdown is performed. The system dump is performed, and the system is rebooted (or halted if the user specifies).

DosShutdown Return Value - ulrc

ulrc (APIRET) - returns
Return code.

DosShutdown returns one of the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
274	ERROR_ALREADY_SHUTDOWN

DosShutdown - Parameters

ShutDownValue (ULONG) - input
Flag indicating the shutdown value.

ShutDownValue has the following states:

<u>Bit</u>	<u>Description</u>
31-16	x00 Reserved.
15-8	Shutdown Type: x00 Return to caller x01 Halt x02 Reboot
7-4	x00 Reserved
3-0	Actions: x00 Flush file system buffers and caches. x01 Flush file system buffers/caches. x02 System dump is requested. The file system will no longer accept write requests.

If a system dump is requested, no shutdown is performed. The system dump is performed, and the system is rebooted (or halted if the user specifies).

ulrc (APIRET) - returns
Return code.

DosShutdown returns one of the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
274	ERROR_ALREADY_SHUTDOWN

DosShutdown - Remarks

DosShutdown can take several minutes to complete its operation; the time depends on the amount of data in the buffers.

If other functions that change file-system data are issued while the system is shut down, either the return code ERROR_ALREADY_SHUTDOWN is set, or the other function calls are blocked permanently.

When DosShutdown reaches a successful completion, the system can be powered-off or restarted.

When *ShutDownValue* equals 1, the system is quiescent at the end of this function request, but the file systems are not locked, and

processing can resume later.

DosShutdown - Related Functions

- [WinShutdownSystem](#)
 - [WinCancelShutdown](#)
-

DosShutdown - Example Code

The following example prepares the file system for the shutdown of the system. Refer to [WinShutdownSystem](#) to perform a complete OS/2 shutdown.

```
#define INCL_DOSFILEMGR
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>

int main(VOID)
{
    APIRET rc = NO_ERROR;    /* Return code */

    rc = DosShutdown(0L);

    if (rc != NO_ERROR) {
        printf("DosShutdown error: return code = %u\n",rc);
        return 1;
    } else {
        printf("The file system has been prepared for shutdown.\n");
    } /* endif */

    return NO_ERROR;
}
```

DosShutdown - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DosStopSession

DosStopSession - Syntax

DosStopSession ends one or all sessions previously started with DosStartSession.

```
#define INCL_dossemgr
#include <os2.h>

ULONG      scope;      /* An indicator that specifies whether only the session specified by idSession, or all se.
ULONG      idSession;  /* The identifier of the session to be ended. */
APIRET     ulrc;       /* Return code. */

ulrc = DosStopSession(scope, idSession);
```

DosStopSession Parameter - scope

scope (ULONG) - input

An indicator that specifies whether only the session specified by *idSession* , or all sessions, should be ended.

Possible values are shown in the following list:

0	STOP_SESSION_SPECIFIED Ends only the specified session.
1	STOP_SESSION_ALL Ends all sessions.

DosStopSession Parameter - idSession

idSession (ULONG) - input

The identifier of the session to be ended.

The value specified for *idSession* must have been returned on a previous call to DosStartSession. *idSession* is ignored if *scope* is equal to STOP_SESSION_ALL.

DosStopSession Return Value - ulrc

ulrc (APIRET) - returns

Return code.

DosStopSession returns one of the following values:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
458	ERROR_SMG_INVALID_STOP_OPTION
459	ERROR_SMG_BAD_RESERVE
463	ERROR_SMG_RETRY_SUB_ALLOC

DosStopSession - Parameters

scope (ULONG) - input

An indicator that specifies whether only the session specified by *idSession* , or all sessions, should be ended.

Possible values are shown in the following list:

0	STOP_SESSION_SPECIFIED Ends only the specified session.
1	STOP_SESSION_ALL Ends all sessions.

idSession (ULONG) - input

The identifier of the session to be ended.

The value specified for *idSession* must have been returned on a previous call to *DosStartSession*. *idSession* is ignored if *scope* is equal to *STOP_SESSION_ALL*.

ulrc (APIRET) - returns

Return code.

DosStopSession returns one of the following values:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
458	ERROR_SMG_INVALID_STOP_OPTION
459	ERROR_SMG_BAD_RESERVE
463	ERROR_SMG_RETRY_SUB_ALLOC

DosStopSession - Remarks

DosStopSession ends one or all sessions previously started with *DosStartSession*.

If the session specified with *DosStopSession* has related sessions, these sessions will also be ended.

If a session is executing in the foreground at the time it is ended, the calling session becomes the foreground session. *DosStopSession* breaks any bond that existed between the calling session and the specified ended session.

The calling session may be executing in either the foreground or background when *DosStopSession* is issued.

Since any process executing in the specified session may refuse to end, the only way to guarantee that the target session has ended is to wait for notification through the termination queue specified with *DosStartSession*.

DosStopSession - Related Functions

- [DosSelectSession](#)
- [DosSetSession](#)
- [DosStartSession](#)

DosStopSession - Example Code

This example starts and stops a child session.

```
#define INCL_DOSPROCESS /* DOS process values - needed for DosSleep */
```

```

#define INCL_DOSSESMGR
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>

int main(VOID) {
    STARTDATA SData      = {0};
    PSZ        PgmTitle   = "This session will stop in a few moments...";
    PgmName     = "CMD.EXE"; /* This starts an OS/2 session */
    APIRET      rc         = NO_ERROR; /* Return code */
    PID         pid        = 0; /* PID returned */
    ULONG       ulSessID   = 0; /* Session ID returned */
    UCHAR       achObjBuf[100] = {0};

    SData.Length = sizeof(STARTDATA);
    SData.Related = SSF_RELATED_CHILD; /* start a dependent session */
    SData.FgBg    = SSF_FGBG_FORE; /* start session in foreground */
    SData.TraceOpt = SSF_TRACEOPT_NONE; /* No trace */
    /* Start an OS/2 session using "CMD.EXE /K" */
    SData.PgmTitle = PgmTitle;
    SData.PgmName = PgmName;
    SData.PgmInputs = "/K"; /* Keep session up */
    SData.TermQ = 0; /* No termination queue */
    SData.Environment = 0; /* No environment string */
    SData.InheritOpt = SSF_INHERTOPT_SHELL; /* Inherit shell's environ. */
    SData.SessionType = SSF_TYPE_WINDOWABLEVIO; /* Windowed VIO session */
    SData.IconFile = 0; /* No icon association */
    SData.PgmHandle = 0;
    SData.PgmControl = SSF_CONTROL_VISIBLE | SSF_CONTROL_MAXIMIZE;
    SData.InitXPos = 30; /* Initial window coordinates */
    SData.InitYPos = 40;
    SData.InitXSize = 200; /* Initial window size */
    SData.InitYSize = 140;
    SData.Reserved = 0;
    SData.ObjectBuffer = achObjBuf; /* Contains info if DosExecPgm fails */
    SData.ObjectBuffLen = (ULONG) sizeof(achObjBuf);

    rc = DosStartSession(&SData, &ulSessID, &pid); /* Start the session */
    if (rc != NO_ERROR) {
        printf ("DosStartSession error : return code = %u\n", rc);
        return 1;
    }

    printf("Waiting a few seconds... then child session will be stopped.\n");
    rc = DosSleep(5000L);

    rc = DosStopSession(STOP_SESSION_SPECIFIED, ulSessID);
    if (rc != NO_ERROR) {
        printf ("DosStopSession error : return code = %u\n", rc);
        return 1;
    }
    return NO_ERROR;
}

```

DosStopSession - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DosTestAddr

DosTestAddr - Syntax

Obtains information about a range of addresses within the virtual-address space of the calling process.

```
#define INCL_DOSMEMMGR
#include <os2.h>

PVOID      pb;      /* The base address of the range of addresses to be queried. */
PULONG     pcb;     /* A pointer to a ULONG containing the size of the range of addresses. */
PULONG     pFlag;   /* A pointer to a ULONG containing a set of attribute flags describing the type of allocation
APIRET     ulrc;    /* Return Code. */

ulrc = DosTestAddr(pb, pcb, pFlag);
```

DosTestAddr Parameter - pb

pb (PVOID) - input
The base address of the range of addresses to be queried.

DosTestAddr Parameter - pcb

pcb (PULONG) - in/out
A pointer to a ULONG containing the size of the range of addresses.

Input	This parameter points to a variable that contains the size, in bytes, of the range of addresses to be queried. The initial value of the variable is rounded to include all addresses addressed by the requested base address and size.
Output	This parameter points to a variable that contains the actual size, in bytes, of the queried range of addresses.

DosTestAddr Parameter - pFlag

pFlag (PULONG) - output
A pointer to a ULONG containing a set of attribute flags describing the type of allocation and access protection for the specified range of addresses.

Allocation Type

PAG_COMMIT (0x00000010)
Pages within the specified region are committed.

PAG_SHARED (0x00002000)
Pages within the specified region are in a shared memory object. Otherwise, the addresses are in a private memory object.

PAG_FREE (0x00004000)

Pages within the specified region are free.

PAG_BASE (0x00010000)

First address in the specified region is the first address in an allocated memory object.

Access Protection

PAG_READ (0x00000001)

Read access to the committed range of addresses is allowed.

PAG_WRITE (0x00000002)

write access to the committed range of addresses is allowed.

PAG_EXECUTE (0x00000004)

Execute access to the committed range of addresses is allowed.

PAG_GUARD (0x00000008)

Access to the committed range of addresses causes a "guard address entered" condition to be raised in the subject process.

DosTestAddr Return Value - ulrc

ulrc (APIRET) - returns
Return Code.

DosTestAddr returns one of the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
487	ERROR_INVALID_ADDRESS

DosTestAddr - Parameters

pb (PVOID) - input
The base address of the range of addresses to be queried.

pcb (PULONG) - in/out
A pointer to a ULONG containing the size of the range of addresses.

Input	This parameter points to a variable that contains the size, in bytes, of the range of addresses to be queried. The initial value of the variable is rounded to include all addresses addressed by the requested base address and size.
-------	--

Output	This parameter points to a variable that contains the actual size, in bytes, of the queried range of addresses.
--------	---

pFlag (PULONG) - output
A pointer to a ULONG containing a set of attribute flags describing the type of allocation and access protection for the specified range of addresses.

Allocation Type

PAG_COMMIT (0x00000010)

Pages within the specified region are committed.

PAG_SHARED (0x00002000)

Pages within the specified region are in a shared memory object. Otherwise, the addresses are in a private memory object.

PAG_FREE (0x00004000)

Pages within the specified region are free.

PAG_BASE (0x00010000)

First address in the specified region is the first address in an allocated memory object.

Access Protection

PAG_READ (0x00000001)

Read access to the committed range of addresses is allowed.

PAG_WRITE (0x00000002)

write access to the committed range of addresses is allowed.

PAG_EXECUTE (0x00000004)

Execute access to the committed range of addresses is allowed.

PAG_GUARD (0x00000008)

Access to the committed range of addresses causes a "guard address entered" condition to be raised in the subject process.

ulrc (APIRET) - returns
Return Code.

DosTestAddr returns one of the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
487	ERROR_INVALID_ADDRESS

DosTestAddr - Remarks

DosTestAddr provides the capability to determine the type and access protection of a range of addresses within the virtual-address space of the subject process. This is the only memory-management function that accepts an address range that is not entirely contained within a previously allocated memory object.

The state of the first address within the region is determined, then subsequent entries in the virtual-address space of the process are scanned from the base address upward until either the entire range of addresses has been scanned, an address with a nonmatching set of attributes is encountered, or the first address in an adjacent allocated memory object is encountered. The region attributes, the length of the range of addresses with matching attributes, and an appropriate error code are returned.

If the entire requested range of addresses does not have a matching set of attributes, then the returned *pcb* parameter value can be used to calculate the address and length of the range of addresses that were not scanned.

Address scanning stops when the first address in an adjacent allocated memory object is encountered. This allows the calling application to determine the appearance of the virtual memory map, including object boundaries.

A region of addresses that is neither committed nor free is considered reserved, that is, it is contained within an allocated memory object but has an access protection of "no access".

If the allocation type returned indicates that the addresses are reserved, that is, neither PAG_COMMIT nor PAG_FREE is set, then the access protection returned is the same as was specified when the object was allocated in the address space of the requesting process.

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

DosTestAddr - Related Functions

- DosSetMem

DosTestAddr - Example Code

This example allocates, commits, queries, uses, and then frees read/write memory.

```
#define INCL_DOSMEMMGR    /* Include DOS Memory Management APIs */
#define INCL_DOSERRORS   /* DOS error values */
#include <os2.h>
#include <stdio.h>
#include <string.h>

int main (VOID)
{
    PVOID  MyObject      = NULL;          /* Pointer to memory object */
    ULONG  ulObjSize     = 0;             /* Size of memory object (in bytes) */
    PULONG pulMemFlags   = 0;             /* Allocation flags for the object */
    PULONG pulMemSize    = 0;             /* Size of memory region for DosSetMem */
    APIRET rc            = NO_ERROR;      /* Return code */

    ulObjSize = 2000;                    /* Will be rounded to an address boundary - 4096 */

    rc = DosAllocMem(&MyObject, /* Pointer to memory object pointer */
                    ulObjSize, /* Size of object to be allocated */
                    PAG_WRITE ); /* Allocate memory as read/writeable */
    if (rc != NO_ERROR) {
        printf("DosAllocMem error: return code = %u\n",rc);
        return 1;
    }
    /* Object can't be used until it is COMMITTED. Since this was
       not done at DosAllocMem time, do it now. */

    rc = DosSetMem(MyObject, /* Pointer to object */
                  ulObjSize, /* Size of area to change */
                  PAG_DEFAULT | PAG_COMMIT ); /* Commit the object */
    if (rc != NO_ERROR) {
        printf("DosSetMem error: return code = %u\n",rc);
        rc = DosFreeMem(MyObject); /* If omitted, OS/2 frees it at termination */
        return 1;
    } else { printf("DosSetMem: complete\n"); }

    strcpy(MyObject, "The memory object has just been used.");

    /* Check COMMIT status of the memory object. */
    pulMemSize = ulObjSize;
    rc = DosQueryMem(MyObject, &pulMemSize, &pulMemFlags);
    if (rc == NO_ERROR) {
        if (pulMemFlags & PAG_COMMIT) {
            printf("Page containing MyObject is now committed.\n");
        } else {
            printf("Error: Page containing MyObject has not been committed.\n");
        } /* endif */
    } else {
        printf("DosQueryMem error: return code = %u\n",rc);
    } /* endif */

    rc = DosFreeMem(MyObject);
    if (rc != NO_ERROR) {
        printf("DosFreeMem error: return code = %u\n",rc);
        return 1;
    }

    return NO_ERROR;
}
```

DosTestAddr - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

DosUnwindException

DosUnwindException - Syntax

DosUnwindException calls and removes exception handlers from a thread's chain of exception handlers.

```
#define INCL_DOSEXCEPTIONS
#include <os2.h>

PEXCEPTIONREGISTRATIONRECORD phandler; /* A pointer to the exception registration record that describes th
PVOID pTargetIP; /* A pointer to where DosUnwindException branches after calling all
PEXCEPTIONREPORTRECORD pERepRec; /* An optional pointer to an exception record. */
LONG ...; /* Optional Parameters. */
APIRET ulrc; /* Return code. */

ulrc = DosUnwindException(phandler, pTargetIP,
    pERepRec, ...);
```

DosUnwindException Parameter - phandler

phandler ([PEXCEPTIONREGISTRATIONRECORD](#)) - input

A pointer to the exception registration record that describes the exception handler to be unregistered.

This parameter can have one of the following values:

Address	A pointer to the exception registration record where the unwind operation should stop.
0	UNWIND_ALL An exit-unwind operation is performed. This removes all exception handlers from the thread, and ends the thread.
-1	END_OF_CHAIN All exception handlers for the thread are unwound.

DosUnwindException Parameter - pTargetIP

pTargetIP (PVOID) - input

A pointer to where DosUnwindException branches after calling all applicable handlers.

DosUnwindException Parameter - pERepRec

pERepRec ([PEXCEPTIONREPORTRECORD](#)) - input
An optional pointer to an exception record.

Set this field to 0 if it is not used.

DosUnwindException Parameter - ...

... (LONG) - input
Optional Parameters.

DosUnwindException can take up to 5 optional parameters containing variables to preserve on the stack. Each parameter, such as a LONG or a ULONG, must be 4 bytes in size.

DosUnwindException Return Value - ulrc

ulrc (APIRET) - returns
Return code.

DosUnwindException returns one of the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION

For a full list of error codes, see *Control Program Programming Reference* .

DosUnwindException - Parameters

phandler ([PEXCEPTIONREGISTRATIONRECORD](#)) - input
A pointer to the exception registration record that describes the exception handler to be unregistered.

This parameter can have one of the following values:

Address	A pointer to the exception registration record where the unwind operation should stop.
0	UNWIND_ALL An exit-unwind operation is performed. This removes all exception handlers from the thread, and ends the thread.
-1	END_OF_CHAIN All exception handlers for the thread are unwound.

pTargetIP (PVOID) - input
A pointer to where DosUnwindException branches after calling all applicable handlers.

pERepRec ([PEXCEPTIONREPORTRECORD](#)) - input
An optional pointer to an exception record.

Set this field to 0 if it is not used.

... (LONG) - input
Optional Parameters.

DosUnwindException can take up to 5 optional parameters containing variables to preserve on the stack. Each parameter, such as a LONG or a ULONG, must be 4 bytes in size.

ulrc (APIRET) - returns
Return code.

DosUnwindException returns one of the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION

For a full list of error codes, see *Control Program Programming Reference* .

DosUnwindException - Remarks

Note:

Do not make Presentation Manager calls from exception handlers.

To preserve variables currently on the stack, pass them as arguments to DosUnwindException using the optional parameters.

DosUnwindException "unwinds" (calls and removes) exception handlers from a thread's chain of registered exception handlers. It can unwind up to, but not including, a specified exception handler, or it can unwind all the exception handlers.

Each exception handler in the linked list from the Thread Information Block (TIB) is called with the unwind bit in the Exception Report Record structure set, indicating an unwind operation. If the call to the exception handler returns, the Exception Registration Record is removed from the linked list, and the next exception handler is processed.

For a detailed list of the system exceptions, see *Control Program Programming Reference* .

DosUnwindException - Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosExitMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler

DosUnwindException - Example Code

The following example unwinds all handlers. It does not resolve the exception, and causes a SYSTEM ERROR pop-up for the exception to be displayed.

```
#define INCL_DOSPROCESS      /* DOS process values (for DosSleep) */
#define INCL_DOSExceptions  /* DOS exception values */
#define INCL_ERRORS         /* DOS error values */
#include <os2.h>
#include <stdio.h>

ULONG _System MyTermHandler( PEXCEPTIONREPORTRECORD p1,
                             PEXCEPTIONREGISTRATIONRECORD p2,
                             PCONTEXTRECORD p3,
                             PVOID pv );
```

```

int main (VOID)
{
    EXCEPTIONREGISTRATIONRECORD RegRec = {0};    /* Exception Registration Record */
    APIRET rc = NO_ERROR; /* Return code */

    /* Add MyTermHandler to this thread's chain of exception handlers */

    RegRec.ExceptionHandler = (ERR)MyTermHandler;
    rc = DosSetExceptionHandler( &RegRec );
    if (rc != NO_ERROR) {
        printf("DosSetExceptionHandler error: return code = %u\n",rc);
        return 1;
    }

    printf("Terminate this program using Ctrl-C or Ctrl-Break.\n");
    printf("You will get a System Error pop-up for the exception.\n");

    rc = DosSleep(60000L); /* Give user plenty of time to comply... */

    rc = DosUnsetExceptionHandler( &RegRec );
    if (rc != NO_ERROR) {
        printf("DosUnsetExceptionHandler error: return code = %u\n",rc);
        return 1;
    }
    return NO_ERROR;
}
/*****
ULONG _System MyTermHandler( PEXCEPTIONREPORTRECORD p1,
                             PEXCEPTIONREGISTRATIONRECORD p2,
                             PCONTEXTRECORD p3,
                             PVOID pv )
{
    APIRET rc = NO_ERROR;

    printf("*** MyTermHandler entered: ExceptionNum = %x\n",p1->ExceptionNum);

    rc = DosUnsetExceptionHandler( p2 ); /* Stop recursive entry to handler */
    rc = DosUnwindException( p2, /* Exception Registration Record */
                             (PVOID) 0, /* BAD ADDRESS!!! */
                             p1); /* Exception Report Record */

    if (rc != NO_ERROR) {
        printf("DosUnwindException error: return code = %u\n", rc);
    }

    return XCPT_CONTINUE_SEARCH;
}

```

DosUnwindException - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

CP Data Types

This section describes the data types that are used with the CP functions, and includes the following data types:

- [CONTEXTRECORD \(POWERPC\)](#)
- [DLLINFO](#)
- [EAOP2](#)

- EXCEPTIONREGISTRATIONRECORD
- EXCEPTIONREPORTRECORD
- FEA2
- FEA2LIST
- FILEFINDBUF3
- FILEFINDBUF4
- FILEFINDBUF13
- FILEFINDBUF14
- FILESTATUS3
- FILESTATUS4
- GEA2
- GEA2LIST
- PROCINFO
- QPROCINFO1
- QPROCINFO2
- QTHREADINFO1
- QTHREADINFO2
- SGID
- STATUSDATA
- THREADINFO

CONTEXTRECORD (POWERPC)

This is the machine specific register contents for the thread at the time of the exception.

```
typedef struct _CONTEXTRECORD (POWERPC) {
    ULONG    ctx_Reg_gpr[32];
    ULONG    ctx_Reg_iar;
    ULONG    ctx_Reg_msr;
    ULONG    ctx_Reg_cr;
    ULONG    ctx_Reg_lr;
    ULONG    ctx_Reg_ctr;
    ULONG    ctx_Reg_xer;
    ULONG    ctx_Reg_mq;
} CONTEXTRECORD (POWERPC);

typedef CONTEXTRECORD (POWERPC) *PCONTEXTRECORD;
```

CONTEXTRECORD (POWERPC) Field - ctx_Reg_gpr[32]

ctx_Reg_gpr[32] (ULONG)
User's general registers.

CONTEXTRECORD (POWERPC) Field - ctx_Reg_iar

ctx_Reg_iar (ULONG)
User's instruction address register.

CONTEXTRECORD (POWERPC) Field - ctx_Reg_msr

ctx_Reg_msr (ULONG)
User's machine state register.

CONTEXTRECORD (POWERPC) Field - ctx_Reg_cr

ctx_Reg_cr (ULONG)
User's condition register.

CONTEXTRECORD (POWERPC) Field - ctx_Reg_lr

ctx_Reg_lr (ULONG)
User's link register.

CONTEXTRECORD (POWERPC) Field - ctx_Reg_ctr

ctx_Reg_ctr (ULONG)
User's count register.

CONTEXTRECORD (POWERPC) Field - ctx_Reg_xer

ctx_Reg_xer (ULONG)
User's storage exception register.

CONTEXTRECORD (POWERPC) Field - ctx_Reg_mq

ctx_Reg_mq (ULONG)
User's mq register.

DLLINFO

There is no description.

```
typedef struct _DLLINFO {
    ULONG      pathname_length;
    UNICHAR     pathname;
} DLLINFO;

typedef DLLINFO *DLLINFO;
```

DLLINFO Field - pathname_length

pathname_length (ULONG)
Length of the path name.

DLLINFO Field - pathname

pathname (UNICHAR)
The pathname of the DLL module.

EAOP2

EAOP2 data structure.

```
typedef struct _EAOP2 {
    PGEA2LIST    fpGEA2List; /* GEA set. */
    PFEA2LIST    fpFEA2List; /* FEA set. */
    ULONG        oError;     /* Offset of FEA error. */
} EAOP2;

typedef EAOP2 *PEAOP2;
```

EAOP2 Field - fpGEA2List

fpGEA2List (PGEA2LIST)
GEA set.

EAOP2 Field - fpFEA2List

fpFEA2List (PFEA2LIST)
FEA set.

EAOP2 Field - oError

oError (ULONG)
Offset of FEA error.

EXCEPTIONREGISTRATIONRECORD

These structures are linked together to form a chain of exception handlers that are dispatched upon receipt of an exception. Exception handlers should not be registered directly from a high level language such as "C". This is the responsibility of the language runtime routine.

```
typedef struct _EXCEPTIONREGISTRATIONRECORD {  
    STRUCT _EXCEPTIONREGISTRATIONRECORD    *prev_structure;    /* Nested exception registration record structure */  
    _ERR                                     *ExceptionHandler; /* Pointer to the ERR function. */  
} EXCEPTIONREGISTRATIONRECORD;  
  
typedef EXCEPTIONREGISTRATIONRECORD *PEXCEPTIONREGISTRATIONRECORD;
```

EXCEPTIONREGISTRATIONRECORD Field - prev_structure

prev_structure (STRUCT _EXCEPTIONREGISTRATIONRECORD *)
Nested exception registration record structure.

This field should be treated as a C-language volatile field. That is, even though this field may be changed in ways unknown to your program, the intent of the original expression will be maintained.

EXCEPTIONREGISTRATIONRECORD Field - ExceptionHandler

ExceptionHandler (_ERR *)
Pointer to the ERR function.

This field must be treated as a C-language volatile field. That is, even though this field may be changed in ways unknown to your program, the intent of the original expression will be maintained.

The ERR function is defined below:

```
typedef ULONG APIENTRY _ERR (PEXCEPTIONREPORTRECORD,  
                             struct _EXCEPTIONREGISTRATIONRECORD *,  
                             PCONTEXTRECORD,  
                             PVOID);  
  
typedef _ERR *ERR;
```

EXCEPTIONREPORTRECORD

This structure contains machine-independent information about an exception or unwind. No system exception will ever have more parameters than the value of EXCEPTION_MAXIMUM_PARAMETERS. User exceptions are not bound to this limit.

```
typedef STRUCT _EXCEPTIONREPORTRECORD {
    ULONG                ExceptionNum;           /* Exception number. */
    ULONG                fHandlerFlags;         /* Handler flags. */
    STRUCT _EXCEPTIONREPORTRECORD *NestedExceptionReportRecord; /* Nested exception report */
    PVOID               ExceptionAddress;       /* Address of the exception */
    ULONG               cParameters;           /* Size of exception specific info */
    ULONG               ExceptionInfo[EXCEPTION_MAXIMUM_PARAMETERS]; /* Exception specific information */
} EXCEPTIONREPORTRECORD;

typedef EXCEPTIONREPORTRECORD *PEXCEPTIONREPORTRECORD;
```

EXCEPTIONREPORTRECORD Field - ExceptionNum

ExceptionNum (ULONG)
Exception number.

EXCEPTIONREPORTRECORD Field - fHandlerFlags

fHandlerFlags (ULONG)
Handler flags.

EXCEPTIONREPORTRECORD Field - NestedExceptionReportRecord

NestedExceptionReportRecord (STRUCT _EXCEPTIONREPORTRECORD *)
Nested exception report record structure.

EXCEPTIONREPORTRECORD Field - ExceptionAddress

ExceptionAddress (PVOID)
Address of the exception.

EXCEPTIONREPORTRECORD Field - cParameters

cParameters (ULONG)
Size of exception specific information.

EXCEPTIONREPORTRECORD Field - ExceptionInfo[EXCEPTION_MAXIMUM_PARAMETERS]

ExceptionInfo[EXCEPTION_MAXIMUM_PARAMETERS] (ULONG)
Exception specific information.

FEA2

32-bit FEA2 data structure.

The data structure defines the format for setting the full extended attributes in the file.

```
typedef struct _FEA2 {  
    ULONG      oNextEntryOffset; /* Offset to next entry. */  
    BYTE       fEA;              /* Extended attributes flag. */  
    BYTE       cbName;           /* Length of szName, not including NULL. */  
    USHORT     cbValue;          /* Value length. */  
    CHAR       szName[1];        /* Extended attribute name. */  
} FEA2;  
  
typedef FEA2 *PFEA2;
```

Extended attributes (EAs) are non-critical by default. A non-critical EA is one that is not necessary to the functionality of the application. If a non-critical EA is lost, the system continues to operate correctly. For example, losing the icons associated with data files does not generally cause any ill effect other than the inability to show the icon.

A *critical extended attribute* is one which is necessary for the correct operation of the operating system or of a particular operation. EAs should be marked as critical if their loss would cause the system or program to perform incorrectly. For example, a mail program might store mail headers in EAs. The loss of the header from a message would normally render the mail program unable to further use that message. This would be unacceptable, so the mail program should mark this EA as critical.

FEA2 Field - oNextEntryOffset

oNextEntryOffset (ULONG)
Offset to next entry.

FEA2 Field - fEA

fEA (BYTE)
Extended attributes flag.

FEA_NEEDEA
Extended attributes are critical.

If this flag is set, this file cannot be copied to a file system that does not support extended attributes. This flag should

only be set if extended attributes are critical to the processing of this file.

0

Extended attributes are not critical.

FEA2 Field - cbName

cbName (BYTE)

Length of *szName* , not including NULL.

This value must be greater than 0.

FEA2 Field - cbValue

cbValue (USHORT)

Value length.

Sending an EA with *cbValue* set to 0 in the FEA2 data structure causes that attribute to be deleted, if possible. Receiving an EA with *cbValue* set to 0 in the FEA2 data structure indicates that the attribute is not present.

FEA2 Field - szName[1]

szName[1] (CHAR)

Extended attribute name.

FEA2LIST

FEA2 data structure.

```
typedef struct _FEA2LIST {  
    ULONG      cbList; /* Total bytes of structure including full list. */  
    FEA2       list[1]; /* Variable-length FEA2 structures. */  
} FEA2LIST;
```

```
typedef FEA2LIST *PFEA2LIST;
```

FEA2LIST Field - cbList

cbList (ULONG)

Total bytes of structure including full list.

FEA2LIST Field - list[1]

list[1] ([FEA2](#))
Variable-length [FEA2](#) structures.

FILEFINDBUF3

32-bit level 1 information (used without EAs).

```
typedef struct _FILEFINDBUF3 {  
    ULONG      oNextEntryOffset; /* Offset of next entry. */  
    FDATE      fdateCreation; /* Date of file creation. */  
    FTIME      ftimeCreation; /* Time of file creation. */  
    FDATE      fdateLastAccess; /* Date of last access. */  
    FTIME      ftimeLastAccess; /* Time of last access. */  
    FDATE      fdateLastWrite; /* Date of last write. */  
    FTIME      ftimeLastWrite; /* Time of last write. */  
    ULONG      cbFile; /* Size of file. */  
    ULONG      cbFileAlloc; /* Allocation size. */  
    ULONG      attrFile; /* File attributes. */  
    UCHAR      cchName;  
    CHAR      achName[CCHMAXPATHCOMP]; /* File name including null terminator. */  
} FILEFINDBUF3;  
  
typedef FILEFINDBUF3 *PFILEFINDBUF3;
```

FILEFINDBUF3 Field - oNextEntryOffset

oNextEntryOffset (ULONG)
Offset of next entry.

FILEFINDBUF3 Field - fdateCreation

fdateCreation (FDATE)
Date of file creation.

FILEFINDBUF3 Field - ftimeCreation

ftimeCreation (FTIME)
Time of file creation.

FILEFINDBUF3 Field - fdateLastAccess

fdateLastAccess (FDATE)
Date of last access.

FILEFINDBUF3 Field - ftimeLastAccess

ftimeLastAccess (FTIME)
Time of last access.

FILEFINDBUF3 Field - fdateLastWrite

fdateLastWrite (FDATE)
Date of last write.

FILEFINDBUF3 Field - ftimeLastWrite

ftimeLastWrite (FTIME)
Time of last write.

FILEFINDBUF3 Field - cbFile

cbFile (ULONG)
Size of file.

FILEFINDBUF3 Field - cbFileAlloc

cbFileAlloc (ULONG)
Allocation size.

FILEFINDBUF3 Field - attrFile

attrFile (ULONG)
File attributes.

FILEFINDBUF3 Field - cchName

cchName (UCHAR)

FILEFINDBUF3 Field - achName[CCHMAXPATHCOMP]

achName[CCHMAXPATHCOMP] (CHAR)
File name including null terminator.

FILEFINDBUF4

32-bit level 2 information (used with EAs).

```
typedef struct _FILEFINDBUF4 {  
    ULONG      oNextEntryOffset; /* Offset of next entry. */  
    FDATE      fdateCreation;     /* Date of file creation. */  
    FTIME      ftimeCreation;     /* Time of file creation. */  
    FDATE      fdateLastAccess;  /* Date of last access. */  
    FTIME      ftimeLastAccess;  /* Time of last access. */  
    FDATE      fdateLastWrite;   /* Date of last write. */  
    FTIME      ftimeLastWrite;   /* Time of last write. */  
    ULONG      cbFile;           /* Size of file. */  
    ULONG      cbFileAlloc;      /* Allocated size. */  
    ULONG      attrFile;         /* File attributes. */  
    ULONG      cbList;           /* Size of the file's extended attributes. */  
    UCHAR      cchName;          /* Length of file name. */  
    CHAR       achName[CCHMAXPATHCOMP]; /* File name including null terminator. */  
} FILEFINDBUF4;  
  
typedef FILEFINDBUF4 *PFILEFINDBUF4;
```

FILEFINDBUF4 Field - oNextEntryOffset

oNextEntryOffset (ULONG)
Offset of next entry.

FILEFINDBUF4 Field - fdateCreation

fdateCreation (FDATE)
Date of file creation.

FILEFINDBUF4 Field - ftimeCreation

ftimeCreation (FTIME)
Time of file creation.

FILEFINDBUF4 Field - fdateLastAccess

fdateLastAccess (FDATE)
Date of last access.

FILEFINDBUF4 Field - ftimeLastAccess

ftimeLastAccess (FTIME)
Time of last access.

FILEFINDBUF4 Field - fdateLastWrite

fdateLastWrite (FDATE)
Date of last write.

FILEFINDBUF4 Field - ftimeLastWrite

ftimeLastWrite (FTIME)
Time of last write.

FILEFINDBUF4 Field - cbFile

cbFile (ULONG)
Size of file.

FILEFINDBUF4 Field - cbFileAlloc

cbFileAlloc (ULONG)
Allocated size.

FILEFINDBUF4 Field - attrFile

attrFile (ULONG)
File attributes.

FILEFINDBUF4 Field - cbList

cbList (ULONG)
Size of the file's extended attributes.

The size is measured in bytes and is the size of the file's entire extended attribute set on the disk.

FILEFINDBUF4 Field - cchName

cchName (UCHAR)
Length of file name.

FILEFINDBUF4 Field - achName[CCHMAXPATHCOMP]

achName[CCHMAXPATHCOMP] (CHAR)
File name including null terminator.

FILEFINDBUF13

32-bit level 1 information (used without EAs).

```
typedef struct _FILEFINDBUF13 {  
    ULONG      oNextEntryOffset; /* Offset of next entry. */  
    FDATE      fdateCreation; /* Date of file creation. */  
    FTIME      ftimeCreation; /* Time of file creation. */  
    FDATE      fdateLastAccess; /* Date of last access. */  
    FTIME      ftimeLastAccess; /* Time of last access. */  
    FDATE      fdateLastWrite; /* Date of last write. */  
    FTIME      ftimeLastWrite; /* Time of last write. */  
    ULONG      cbFile; /* Size of file. */  
    ULONG      cbFileAlloc; /* Allocation size. */  
    ULONG      attrFile; /* File attributes. */  
    ULONG      position; /* Position to continue the search from. */  
    UCHAR      cchName;  
    CHAR      achName[CCHMAXPATHCOMP]; /* File name including null terminator. */  
} FILEFINDBUF13;  
  
typedef FILEFINDBUF13 *PFILEFINDBUF13;
```

FILEFINDBUF13 Field - oNextEntryOffset

oNextEntryOffset (ULONG)
Offset of next entry.

FILEFINDBUF13 Field - fdateCreation

fdateCreation (FDATE)
Date of file creation.

FILEFINDBUF13 Field - ftimeCreation

ftimeCreation (FTIME)
Time of file creation.

FILEFINDBUF13 Field - fdateLastAccess

fdateLastAccess (FDATE)
Date of last access.

FILEFINDBUF13 Field - ftimeLastAccess

ftimeLastAccess (FTIME)
Time of last access.

FILEFINDBUF13 Field - fdateLastWrite

fdateLastWrite (FDATE)
Date of last write.

FILEFINDBUF13 Field - ftimeLastWrite

ftimeLastWrite (FTIME)
Time of last write.

FILEFINDBUF13 Field - cbFile

cbFile (ULONG)
Size of file.

FILEFINDBUF13 Field - cbFileAlloc

cbFileAlloc (ULONG)
Allocation size.

FILEFINDBUF13 Field - attrFile

attrFile (ULONG)
File attributes.

FILEFINDBUF13 Field - position

position (ULONG)
Position to continue the search from.

FILEFINDBUF13 Field - cchName

cchName (UCHAR)

FILEFINDBUF13 Field - achName[CCHMAXPATHCOMP]

achName[CCHMAXPATHCOMP] (CHAR)
File name including null terminator.

FILEFINDBUF14

32-bit level 2 information (used with EAs).

```
typedef struct _FILEFINDBUF14 {  
    ULONG      oNextEntryOffset; /* Offset of next entry. */  
    FDATE      fdateCreation; /* Date of file creation. */  
    FTIME      ftimeCreation; /* Time of file creation. */  
    FDATE      fdateLastAccess; /* Date of last access. */  
    FTIME      ftimeLastAccess; /* Time of last access. */  
    FDATE      fdateLastWrite; /* Date of last write. */  
    FTIME      ftimeLastWrite; /* Time of last write. */  
    ULONG      cbFile; /* Size of file. */  
    ULONG      cbFileAlloc; /* Allocated size. */  
    ULONG      attrFile; /* File attributes. */  
    ULONG      position; /* Position to continue the search from. */  
    ULONG      cbList; /* Size of the file's extended attributes. */  
    UCHAR      cchName; /* Length of file name. */  
    CHAR      achName[CCHMAXPATHCOMP]; /* File name including null terminator. */  
} FILEFINDBUF14;  
  
typedef FILEFINDBUF14 *PFILEFINDBUF14;
```

FILEFINDBUF14 Field - oNextEntryOffset

oNextEntryOffset (ULONG)
Offset of next entry.

FILEFINDBUF14 Field - fdateCreation

fdateCreation (FDATE)
Date of file creation.

FILEFINDBUF14 Field - ftimeCreation

ftimeCreation (FTIME)
Time of file creation.

FILEFINDBUF14 Field - fdateLastAccess

fdateLastAccess (FDATE)
Date of last access.

FILEFINDBUF14 Field - ftimeLastAccess

ftimeLastAccess (FTIME)
Time of last access.

FILEFINDBUF14 Field - fdateLastWrite

fdateLastWrite (FDATE)
Date of last write.

FILEFINDBUF14 Field - ftimeLastWrite

ftimeLastWrite (FTIME)
Time of last write.

FILEFINDBUF14 Field - cbFile

cbFile (ULONG)

Size of file.

FILEFINDBUF14 Field - cbFileAlloc

cbFileAlloc (ULONG)
Allocated size.

FILEFINDBUF14 Field - attrFile

attrFile (ULONG)
File attributes.

FILEFINDBUF14 Field - position

position (ULONG)
Position to continue the search from.

FILEFINDBUF14 Field - cbList

cbList (ULONG)
Size of the file's extended attributes.

The size is measured in bytes and is the size of the file's entire extended attribute set on the disk.

FILEFINDBUF14 Field - cchName

cchName (UCHAR)
Length of file name.

FILEFINDBUF14 Field - achName[CCHMAXPATHCOMP]

achName[CCHMAXPATHCOMP] (CHAR)
File name including null terminator.

FILESTATUS3

32-bit level 1 (FIL_STANDARD) information.

```
typedef struct _FILESTATUS3 {  
    FDATE      fdateCreation; /* Date of file creation. */  
    FTIME      ftimeCreation; /* Time of file creation. */  
    FDATE      fdateLastAccess; /* Date of last access. */  
    FTIME      ftimeLastAccess; /* Time of last access. */  
    FDATE      fdateLastWrite; /* Date of last write. */  
    FTIME      ftimeLastWrite; /* Time of last write. */  
    ULONG      cbFile; /* File size (end of data). */  
    ULONG      cbFileAlloc; /* File allocated size. */  
    ULONG      attrFile; /* Attributes of the file. */  
} FILESTATUS3;
```

```
typedef FILESTATUS3 *PFILESTATUS3;
```

FILESTATUS3 Field - fdateCreation

fdateCreation (FDATE)
Date of file creation.

FILESTATUS3 Field - ftimeCreation

ftimeCreation (FTIME)
Time of file creation.

FILESTATUS3 Field - fdateLastAccess

fdateLastAccess (FDATE)
Date of last access.

FILESTATUS3 Field - ftimeLastAccess

ftimeLastAccess (FTIME)
Time of last access.

FILESTATUS3 Field - fddateLastWrite

fddateLastWrite (FDATE)
Date of last write.

FILESTATUS3 Field - ftimeLastWrite

ftimeLastWrite (FTIME)
Time of last write.

FILESTATUS3 Field - cbFile

cbFile (ULONG)
File size (end of data).

FILESTATUS3 Field - cbFileAlloc

cbFileAlloc (ULONG)
File allocated size.

FILESTATUS3 Field - attrFile

attrFile (ULONG)
Attributes of the file.

FILESTATUS4

32-bit level 2 (FIL_QUERYEASIZE) information.

```
typedef struct _FILESTATUS4 {  
    FDATE      fddateCreation;    /* Date of file creation. */  
    FTIME      ftimeCreation;     /* Time of file creation. */  
    FDATE      fddateLastAccess;  /* Date of last access. */  
    FTIME      ftimeLastAccess;   /* Time of last access. */  
    FDATE      fddateLastWrite;   /* Date of last write. */  
    FTIME      ftimeLastWrite;    /* Time of last write. */  
};
```

```
    ULONG      cbFile;          /* File size (end of data). */
    ULONG      cbFileAlloc;     /* File allocated size. */
    ULONG      attrFile;        /* Attributes of the file. */
    ULONG      cbList;          /* Length of entire EA set. */
} FILESTATUS4;

typedef FILESTATUS4 *PFILESTATUS4;
```

FILESTATUS4 Field - fddateCreation

fddateCreation (FDATE)
Date of file creation.

FILESTATUS4 Field - ftimeCreation

ftimeCreation (FTIME)
Time of file creation.

FILESTATUS4 Field - fddateLastAccess

fddateLastAccess (FDATE)
Date of last access.

FILESTATUS4 Field - ftimeLastAccess

ftimeLastAccess (FTIME)
Time of last access.

FILESTATUS4 Field - fddateLastWrite

fddateLastWrite (FDATE)
Date of last write.

FILESTATUS4 Field - ftimeLastWrite

ftimeLastWrite (FTIME)
Time of last write.

FILESTATUS4 Field - cbFile

cbFile (ULONG)
File size (end of data).

FILESTATUS4 Field - cbFileAlloc

cbFileAlloc (ULONG)
File allocated size.

FILESTATUS4 Field - attrFile

attrFile (ULONG)
Attributes of the file.

FILESTATUS4 Field - cbList

cbList (ULONG)
Length of entire EA set.

GEA2

32-bit Level 3 (FIL_QUERYEASFROMLIST) File Information - Get Extended Attributes.

```
typedef struct _GEA2 {
    ULONG      oNextEntryOffset; /* Offset to next entry. */
    BYTE       cbName;           /* Name length not including NULL. */
    CHAR       szName[1];        /* Attribute name. */
} GEA2;

typedef GEA2 *PGEA2;
```

GEA2 Field - oNextEntryOffset

oNextEntryOffset (ULONG)
Offset to next entry.

GEA2 Field - cbName

cbName (BYTE)
Name length not including NULL.

GEA2 Field - szName[1]

szName[1] (CHAR)
Attribute name.

GEA2LIST

Get Extended Attributes list.

```
typedef struct _GEA2LIST {  
    ULONG      cbList; /* Total bytes of structure including full list. */  
    GEA2       list[1]; /* Variable-length GEA2 structures. */  
} GEA2LIST;  
  
typedef GEA2LIST *PGEA2LIST;
```

GEA2LIST Field - cbList

cbList (ULONG)
Total bytes of structure including full list.

GEA2LIST Field - list[1]

list[1] ([GEA2](#))
Variable-length GEA2 structures.

HMODULE

Module handle.

```
typedef LHANDLE HMODULE;
```

PROCINFO

```
typedef struct _PROCINFO {
    ULONG        packet_size;
    ULONG        packet_revision_no;
    ULONG        parent_process_ID;
    ULONG        process_path_name_length;
    UNICHAR      process_path_name;
    ULONG        process_status;
    ULONG        process_status;
    ULONG        sgid;
    ULONG        no_of_runtime_linked_dlls;
    ULONG        runtime_dll_info_length;
    ULONG        no_of_threads;
    PVOID        thread_buffer;
    PVOID        dll_info;
} PROCINFO;

typedef PROCINFO *PROCINFO;
```

PROCINFO Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

PROCINFO Field - packet_revision_no

packet_revision_no (ULONG)
The revision number of this structure template.

PROCINFO Field - parent_process_ID

parent_process_ID (ULONG)
The ID of the parent process.

PROCINFO Field - process_path_name_length

process_path_name_length (ULONG)
Length of the process path-name parameter.

PROCINFO Field - process_path_name

process_path_name (UNICHAR)
The process path name.

PROCINFO Field - process_status

process_status (ULONG)
The process type.

1	PNS
2	MVM
3	OS/2 personality

PROCINFO Field - process_status

process_status (ULONG)
The status of the process.

1	Doing Exitlist processing
2	Exiting thread 1.
3	The whole process is exiting.
4	Parent care about termination.
5	Parent did an exec and wait.
6	Process is dying.
7	Process is in embryonic state.

PROCINFO Field - sgid

sgid (ULONG)
Process screen group.

PROCINFO Field - no_of_runtime_linked_dlls

no_of_runtime_linked_dlls (ULONG)
Number of runtime-linked DLLs.

PROCINFO Field - runtime_dll_info_length

runtime_dll_info_length (ULONG)
Length required to describe the path names of runtime DLLs. If only summary is requested, length will be set to 0.

PROCINFO Field - no_of_threads

no_of_threads (ULONG)
The total number of threads.

PROCINFO Field - thread_buffer

thread_buffer (PVOID)
Threads information. All the threads information will be kept here. The thread information can be accessed by using THREADINFO no_of_threads times. If only summary is required, this information will not be returned.

PROCINFO Field - dll_info

dll_info (PVOID)
The linked DLL's path names. All the DLL path names can be retrieved by using DLLINFO no_of_runtime_linked DLLs times. The total space used for dll_info will be indicated in runtime_dll_info_length parameter. If only summary is requested, this information will not be returned.

QPROCINFO1

```

typedef struct _QPROCINFO1 {
    ULONG      QPI_length;          /* Type of process. */
    ULONG      QPI_type;            /* Process or task ID. */
    ULONG      QPI_pid;             /* Parent process or task ID. */
    ULONG      QPI_parentpid;       /* Session identifier. */
    SID        QPI_sid;             /* Session type. */
    ULONG      QPI_sessiontype;     /* Process priority. */
    ULONG      QPI_priority;        /* Process flags. */
    ULONG      QPI_flags;           /* User identifier in UUID format. */
    ULONG      QPI_user;            /* Group identifier in UUID format. */
    ULONG      QPI_group;           /* Module handle. */
    ULONG      QPI_hmodule;         /* Offset to the process name. */
    ULONG      QPI_nameoff;
} QPROCINFO1;

typedef QPROCINFO1 *QPROCINFO1;

```

QPROCINFO1 Field - QPI_length

QPI_length (ULONG)
Length of this entry data. Except for the last entry, this is offset in bytes to the next entry.

QPROCINFO1 Field - QPI_type

QPI_type (ULONG)
Type of process.

May be one of the following types:

QPI_OTHER	Non OS/2 task
QPI_OS2	OS/2 process

QPROCINFO1 Field - QPI_pid

QPI_pid (ULONG)
Process or task ID.

For OS/2 processes it will be the OS/2 process id. For others it will be the task id.

QPROCINFO1 Field - QPI_parentpid

QPI_parentpid (ULONG)

Parent process or task ID.

For OS/2 processes it will be OS/2 process id. For others it will be the task id.

QPROCINFO1 Field - QPI_sid

QPI_sid (SID)

Session identifier.

Information will be returned only for OS/2 processes. For other tasks, the session ID will be set to zero.

QPROCINFO1 Field - QPI_sessiontype

QPI_sessiontype (ULONG)

Session type.

Information will be returned only for OS/2 processes. For other tasks, the session type will be set to zero.

QPROCINFO1 Field - QPI_priority

QPI_priority (ULONG)

Process priority.

Only OS/2 processes priority information is returned. Non-OS/2 tasks priority will be set zero.

QPROCINFO1 Field - QPI_flags

QPI_flags (ULONG)

Process flags.

Process flags and their meaning as follows:

ID	Value	Meaning
QPI_FLAG_ASYNC	0x0001	Asynchronous
QPI_FLAG_RESULT	0x0002	Keep results
QPI_FLAG_DEBUG	0x0004	Under debug
QPI_FLAG_ZOMBIE	0x0008	Zombie process
QPI_FLAG_ENDING	0x0010	Process is ending
QPI_FLAG_PRIV	0x0080	Process has

QPROCINFO1 Field - QPI_user

QPI_user (ULONG)
User identifier in UUID format.

QPROCINFO1 Field - QPI_group

QPI_group (ULONG)
Group identifier in UUID format.

QPROCINFO1 Field - QPI_hmodule

QPI_hmodule (ULONG)
Module handle.

Information will be returned only for OS/2 processes.

QPROCINFO1 Field - QPI_nameoff

QPI_nameoff (ULONG)
Offset to the process name.

The offset is calculated from the beginning of InfoBuffer. The process or task name is a null-terminated ASCII string.

QPROCINFO2

```
typedef struct _QPROCINFO2 {
    ULONG    QPI_length;
    ULONG    QPI_type;          /* Type of process. */
    ULONG    QPI_pid;          /* Process or task ID. */
    ULONG    QPI_parentpid;    /* Parent process or task ID. */
    SID      QPI_sid;          /* Session identifier. */
    ULONG    QPI_sessiontype;  /* Session type. */
    ULONG    QPI_priority;     /* Process priority. */
    ULONG    QPI_flags;        /* Process flags. */
}
```

```

ULONG      QPI_user;          /* User identifier in UUID format. */
ULONG      QPI_group;         /* Group identifier in UUID format. */
ULONG      QPI_hmodule;       /* Module handle. */
ULONG      QPI_nameoff;       /* Offset to the process name. */
ULONG      QPI_threadcount;    /* Thread count for the process. */
ULONG      QPI_libcount;       /* Count of libraries. */
ULONG      QPI_liboff;        /* Offset to libraries. */
} QPROCINFO2;

typedef QPROCINFO2 *QPROCINFO2;

```

QPROCINFO2 Field - QPI_length

QPI_length (ULONG)
Length of this entry data. Except for the last entry, this is offset in bytes to the next entry.

QPROCINFO2 Field - QPI_type

QPI_type (ULONG)
Type of process.

May be one of the following types:

- | | |
|-----------|---------------|
| QPI_OTHER | Non OS/2 task |
| QPI_OS2 | OS/2 process |

QPROCINFO2 Field - QPI_pid

QPI_pid (ULONG)
Process or task ID.

For OS/2 processes it will be the OS/2 process id. For others it will be the task id.

QPROCINFO2 Field - QPI_parentpid

QPI_parentpid (ULONG)
Parent process or task ID.

For OS/2 processes it will be OS/2 process id. For others it will be the task id.

QPROCINFO2 Field - QPI_sid

QPI_sid (SID)
Session identifier.

Information will be returned only for OS/2 processes. For other tasks, the session ID will be set to zero.

QPROCINFO2 Field - QPI_sessiontype

QPI_sessiontype (ULONG)
Session type.

Information will be returned only for OS/2 processes. For other tasks, the session type will be set to zero.

QPROCINFO2 Field - QPI_priority

QPI_priority (ULONG)
Process priority.

Both OS/2 processes and other tasks priority information are returned.

QPROCINFO2 Field - QPI_flags

QPI_flags (ULONG)
Process flags.

Process flags and their meaning as follows:

ID	Value	Meaning
QPI_FLAG_ASYNC	0x0001	Asynchronous
QPI_FLAG_RESULT	0x0002	Keep results
QPI_FLAG_DEBUG	0x0004	Under debug
QPI_FLAG_ZOMBIE	0x0008	Zombie process
QPI_FLAG_ENDING	0x0010	Process is ending
QPI_FLAG_PRIV	0x0080	Process has privilege

QPROCINFO2 Field - QPI_user

QPI_user (ULONG)
User identifier in UUID format.

QPROCINFO2 Field - QPI_group

QPI_group (ULONG)
Group identifier in UUID format.

QPROCINFO2 Field - QPI_hmodule

QPI_hmodule (ULONG)
Module handle.

Information will be returned only for OS/2 processes.

QPROCINFO2 Field - QPI_nameoff

QPI_nameoff (ULONG)
Offset to the process name.

The offset is calculated from the beginning of InfoBuffer. The process or task name is a null-terminated ASCII string.

QPROCINFO2 Field - QPI_threadcount

QPI_threadcount (ULONG)
Thread count for the process.

QPROCINFO2 Field - QPI_libcount

QPI_libcount (ULONG)
Count of libraries.

QPROCINFO2 Field - QPI_liboff

QPI_liboff (ULONG)
Offset to libraries.

Offset is calculated from the beginning of theInfoBuffer. The libraries list is returned as an array of HMODULE entries. DosQueryModuleName can be used to return the name of the library.

QTHREADINFO1

```
typedef struct _QTHREADINFO1 {
    ULONG      QTI_length;
    ULONG      QTI_tid;      /* Thread ID. */
    ULONG      QTI_ordinal;  /* Thread ordinal number. */
    ULONG      QTI_tib;      /* Task info block. */
    ULONG      QTI_priority; /* Thread priority. */
} QTHREADINFO1;

typedef QTHREADINFO1 *QTHREADINFO1;
```

QTHREADINFO1 Field - QTI_length

QTI_length (ULONG)
Length of this entry data. Except for the last entry, this is offset in bytes to the next entry.

QTHREADINFO1 Field - QTI_tid

QTI_tid (ULONG)
Thread ID.

For non-OS/2 tasks, tid and ordinal are the same.

QTHREADINFO1 Field - QTI_ordinal

QTI_ordinal (ULONG)
Thread ordinal number.

For non-OS/2 tasks, tid and ordinal are the same.

QTHREADINFO1 Field - QTI_tib

QTI_tib (ULONG)
Task info block.

The TIB address is only valid in the address space of the task containing the thread.

QTHREADINFO1 Field - QTI_priority

QTI_priority (ULONG)
Thread priority.

Only OS/2 process thread priority information is returned.

QTHREADINFO2

```
typedef struct _QTHREADINFO2 {
    ULONG    QTI_length;
    ULONG    QTI_tid;           /* Thread ID. */
    ULONG    QTI_ordinal;       /* Thread ordinal number. */
    ULONG    QTI_tib;           /* Task info block. */
    ULONG    QTI_priority;       /* Thread priority. */
    ULONG    QTI_maxpriority;    /* Maximum priority. */
    ULONG    QTI_sleep;         /* Sleep time in seconds. */
    ULONG    QTI_suspendcount;   /* Suspend count. */
    ULONG    QTI_state;         /* Thread state. */
    ULONG    QTI_flags;         /* Thread flags. */
    ULONG    QTI_wait;          /* Wait value. */
    ULONG    QTI_usertime;       /* User time in milliseconds. */
    ULONG    QTI_systemtime;     /* System time in milliseconds. */
} QTHREADINFO2;

typedef QTHREADINFO2 *QTHREADINFO2;
```

QTHREADINFO2 Field - QTI_length

QTI_length (ULONG)
Length of this entry data. Except for the last entry, this is offset in bytes to the next entry.

QTHREADINFO2 Field - QTI_tid

QTI_tid (ULONG)
Thread ID.

For non-OS/2 tasks, tid and ordinal are the same.

QTHREADINFO2 Field - QTI_ordinal

QTI_ordinal (ULONG)
Thread ordinal number.

For non-OS/2 tasks, tid and ordinal are the same.

QTHREADINFO2 Field - QTI_tib

QTI_tib (ULONG)
Task info block.

The TIB address is only valid in the address space of the task containing the thread.

QTHREADINFO2 Field - QTI_priority

QTI_priority (ULONG)
Thread priority.

Only OS/2 process thread priority information is returned.

QTHREADINFO2 Field - QTI_maxpriority

QTI_maxpriority (ULONG)
Maximum priority.

QTHREADINFO2 Field - QTI_sleep

QTI_sleep (ULONG)
Sleep time in seconds.

QTHREADINFO2 Field - QTI_suspendcount

QTI_suspendcount (ULONG)

Suspend count.

QTHREADINFO2 Field - QTI_state

QTI_state (ULONG)

Thread state.

May be one of the following values:

QTI_STATE_RUNNING	Running normally
QTI_STATE_STOPPED	Stopped
QTI_STATE_WAITING	Waiting normally
QTI_STATE_UNINTERRUPTIBLE	In an uninterruptible state
QTI_STATE_HALTED	Halted at a clean point

QTHREADINFO2 Field - QTI_flags

QTI_flags (ULONG)

Thread flags.

The following flags are defined:

QTI_FLAGS_SWAPPED (0x0001)	Thread is swapped out
QTI_FLAGS_IDLE (0x0002)	Thread is an idle thread

QTHREADINFO2 Field - QTI_wait

QTI_wait (ULONG)

Wait value.

If the thread wait value is unknown zero is returned.

QTHREADINFO2 Field - QTI_usertime

QTI_usertime (ULONG)

User time in milliseconds.

QTHREADINFO2 Field - QTI_systemtime

QTI_systemtime (ULONG)

System time in milliseconds.

SGID

32-bit value used as a session identifier.

```
typedef USHORT SGID;
```

STATUSDATA

Status data structure.

```
typedef struct _STATUSDATA {
    USHORT    Length;        /* The length of the data structure, in bytes, including Length itself. */
    USHORT    SelectInd;     /* An indicator that specifies whether the target session should be flagged as selectable. */
    USHORT    BondInd;       /* An indicator that specifies which session to bring to the foreground the next time. */
} STATUSDATA;

typedef STATUSDATA *PSTATUSDATA;
```

STATUSDATA Field - Length

Length (USHORT)

The length of the data structure, in bytes, including *Length* itself.

STATUSDATA Field - SelectInd

SelectInd (USHORT)

An indicator that specifies whether the target session should be flagged as selectable or non-selectable.

Possible values are shown in the following list:

0	SET_SESSION_UNCHANGED
	Leaves the current setting unchanged.

- | | |
|---|--|
| 1 | SET_SESSION_SELECTABLE
Makes the target session selectable. |
| 2 | SET_SESSION_NON_SELECTABLE
Makes the target session non-selectable. A non-selectable session is not selectable from the Shell switch list, nor can the user jump to it via the system hot key. The operator may continue to select a non-selectable windowed session by pressing a mouse button within a visible part of the window. |

STATUSDATA Field - BondInd

BondInd (USHORT)

An indicator that specifies which session to bring to the foreground the next time the parent session is selected.

Possible values are shown in the following list:

- | | |
|---|--|
| 0 | SET_SESSION_UNCHANGED
Leaves the current setting unchanged. |
| 1 | SET_SESSION_BOND
Establishes a bond between the parent session and the child session. The child session is brought to the foreground the next time the parent session is selected. If the child session is selected, the child session is brought to the foreground. |
| 2 | SET_SESSION_NO_BOND
Specifies bringing the parent session to the foreground the next time the parent session is selected, and bringing the child session to the foreground if the child is selected. Any bond previously established with the child session specified is broken. |

THREADINFO

```
typedef struct _THREADINFO {
    ULONG      thread_id;
    ULONG      sleepid;
    ULONG      thread_priority;
    ULONG      thread_system_time;
    ULONG      thread_user_time;
    ULONG      thread_state;
} THREADINFO;

typedef THREADINFO *THREADINFO;
```

THREADINFO Field - thread_id

thread_id (ULONG)
ID of the thread.

THREADINFO Field - sleepid

sleepid (ULONG)
Sleep ID thread is sleeping on.

THREADINFO Field - thread_priority

thread_priority (ULONG)
Priority of the thread.

THREADINFO Field - thread_system_time

thread_system_time (ULONG)
Thread system time.

THREADINFO Field - thread_user_time

thread_user_time (ULONG)
Thread user time.

THREADINFO Field - thread_state

thread_state (ULONG)
State of the thread.

RAM Semaphore Support

RAM semaphores are fast semaphores designed for minimum function and maximum performance. They should be used to protect a data object when a conflict is unlikely.

The RAM semaphore uses a small area of memory that is available to all users of the RAM Semaphore. The internal structure of this memory is defined only to the implementation. No use of them can be made by others.

The following is an example RAM semaphore structure.

```
typedef struct {  
    ULONG semwords[8];          /* Semaphore private memory */  
} RAMSEM, *PRAMSEM;
```

The chapter includes the following sections:

- [RAM Semaphore Functions](#)
- [RAM Semaphore Support Data Types](#)

RAM Semaphore Functions

This section describes the following RAM Semaphore functions:

- [DosCloseRamSem](#)
- [DosOpenRamSem](#)
- [DosReleaseRamSem](#)
- [DosRequestRamSem](#)

DosCloseRamSem

DosCloseRamSem - Syntax

DosCloseRamSem detaches a RAM semaphore from this process.

```
#define INCL_DOSSEMAPHORES
#include <os2.h>

PRAMSEM    pramsem; /* Pointer to the RAM semaphore structure. */
APIRET     rc;      /* Return code. */

rc = DosCloseRamSem(pramsem);
```

DosCloseRamSem Parameter - pramsem

pramsem ([PRAMSEM](#)) - in/out
Pointer to the RAM semaphore structure.

DosCloseRamSem Return Value - rc

rc (APIRET) - returns
Return code.

DosCloseRamSem returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

DosCloseRamSem - Parameters

pramsem ([PRAMSEM](#)) - in/out
Pointer to the RAM semaphore structure.

rc (APIRET) - returns
Return code.

DosCloseRamSem returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

DosCloseRamSem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

DosOpenRamSem

DosOpenRamSem - Syntax

DosOpenRamSem attaches a RAM semaphore to this process. The semaphore is opened for all threads in the calling process.

```
#define INCL_DOSSEMAPHORES
#include <os2.h>

PRAMSEM    pramsem; /* Pointer to the RAM semaphore structure. */
APIRET     rc;       /* Return code. */

rc = DosOpenRamSem(pramsem);
```

DosOpenRamSem Parameter - pramsem

pramsem ([PRAMSEM](#)) - in/out
Pointer to the RAM semaphore structure.

DosOpenRamSem Return Value - rc

rc (APIRET) - returns
Return code.

DosOpenRamSem returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

DosOpenRamSem - Parameters

pramsem ([PRAMSEM](#)) - in/out
Pointer to the RAM semaphore structure.

rc (APIRET) - returns
Return code.

DosOpenRamSem returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

DosOpenRamSem - Remarks

DosOpenRamSem attaches a RAM semaphore for use by this process. The semaphore is opened for all threads in the calling process.

DosOpenRamSem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DosReleaseRamSem

DosReleaseRamSem - Syntax

DosReleaseRamSem releases a RAM semaphore. This is used at the exit from a range of code which must not be executed simultaneously in multiple threads.

```
#define INCL_DOSSEMAPHORES
#include <os2.h>

PRAMSEM    pramsem; /* Pointer to the RAM semaphore structure. */
ULONG      flag;    /* Release flag. */
APIRET     rc;      /* Return code. */

rc = DosReleaseRamSem(pramsem, flag);
```

DosReleaseRamSem Parameter - pramsem

pramsem (**PRAMSEM**) - in/out
Pointer to the RAM semaphore structure.

DosReleaseRamSem Parameter - flag

flag (ULONG) - input
Release flag.

The release flag may have the following values:

0x00000001	SEM_RELEASE_ALL
0x00000002	SEM_RELEASE_UNOWNED
0x00000004	SEM_RELEASE_PID_OWNED

DosReleaseRamSem Return Value - rc

rc (APIRET) - returns
Return code.

DosReleaseRamSem returns one of the following values:

0	NO_ERROR
288	ERROR_NOT_OWNER

DosReleaseRamSem - Parameters

pramsem ([PRAMSEM](#)) - in/out
Pointer to the RAM semaphore structure.

flag (ULONG) - input
Release flag.

The release flag may have the following values:

0x00000001	SEM_RELEASE_ALL
0x00000002	SEM_RELEASE_UNOWNED
0x00000004	SEM_RELEASE_PID_OWNED

rc (APIRET) - returns
Return code.

DosReleaseRamSem returns one of the following values:

0	NO_ERROR
288	ERROR_NOT_OWNER

DosReleaseRamSem - Remarks

DosReleaseRamSem is used at the end of a section of code protected by a [DosRequestRamSem](#). It is valid only when the semaphore is owned by this thread.

DosReleaseRamSem decrements the semaphore-use count. If the use count becomes 0, ownership of the semaphore is given up, and any threads waiting for the semaphore are released.

DosReleaseRamSem is a functional replacement for OS/2 1.x DosFSRamSemClear.

DosReleaseRamSem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DosRequestRamSem

DosRequestRamSem - Syntax

DosRequestRamSem obtains a RAM semaphore. This is used at the entry to a range of code that must not be executed simultaneously in multiple threads.

```
#define INCL_DOSSEMAPHORES
#include <os2.h>

PRAMSEM    pramsem; /* Pointer to the RAM semaphore structure. */
LONG       timeout;
APIRET     rc;      /* Return code. */

rc = DosRequestRamSem(pramsem, timeout);
```

DosRequestRamSem Parameter - pramsem

pramsem (**PRAMSEM**) - in/out
Pointer to the RAM semaphore structure.

DosRequestRamSem Parameter - timeout

timeout (LONG) - input
The number of milliseconds to wait for the semaphore to be released before returning with an error:

-1	Wait indefinitely.
0	Do not wait; return immediately if the semaphore is in use.
>0	Number of milliseconds to wait if the semaphore is in use.

DosRequestRamSem Return Value - rc

rc (APIRET) - returns
Return code.

DosRequestRamSem returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
121	ERROR_SEM_TIMEOUT

DosRequestRamSem - Parameters

pramsem ([PRAMSEM](#)) - in/out
Pointer to the RAM semaphore structure.

timeout (LONG) - input
The number of milliseconds to wait for the semaphore to be released before returning with an error:

-1	Wait indefinitely.
0	Do not wait; return immediately if the semaphore is in use.
>0	Number of milliseconds to wait if the semaphore is in use.

rc (APIRET) - returns
Return code.

DosRequestRamSem returns one of the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
121	ERROR_SEM_TIMEOUT

DosRequestRamSem - Remarks

DosRequestRamSem provides a multiprocessor-safe RAM semaphore mechanism. When DosRequestRamSem is called, if the semaphore is not in use, this thread becomes the owner of the semaphore, and the use count is incremented. If this thread is already the owner of the semaphore, the use count is incremented. If another thread owns the semaphore, this thread waits until the owning thread releases the semaphore.

If a positive timeout value is given and the semaphore cannot be obtained within the specified time, an error is returned.

DosRequestRamSem is normally used to protect a section of code or object from being acted on by multiple threads at the same time. When the thread that called DosRequestRamSem is through using the object, it calls [DosReleaseRamSem](#), which decrements the use count. The use-count mechanism allows the thread to do recursive semaphore requests.

DosRequestRamSem can be used by multiple threads within a single process or by threads in separate processes. If threads in separate processes wish to use the RAM semaphore, the RAM semaphore must be in shared memory. Before the first DosRequestRamSem is done, the memory of the semaphore should be set to 0. This memory is subsequently used by DosRequestRamSem and [DosReleaseRamSem](#) and should not be modified.

RAM semaphores should be used to protect a small section of code or memory, when the expectation is that no wait is necessary. The time for the no-wait case is very small, but the wait times might be longer than they are when using Event or Mutex semaphores.

DosRequestRamSem is a functional replacement for OS/2 1.x DosFSRamSemRequest.

DosRequestRamSem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

RAM Semaphore Support Data Types

This section describes the following data types that are used with the RAM Semaphore functions:

- [RAMSEM](#)

RAMSEM

RAM Semaphore structure.

```
typedef struct _RAMSEM {  
    ULONG      semwords[8]; /* Semaphore private memory. */  
} RAMSEM;  
  
typedef RAMSEM *PRAMSEM;
```

RAMSEM Field - semwords[8]

semwords[8] (ULONG)
Semaphore private memory.

Network

This chapter describes all of the network functions and data types provided in the IBM Developer's Toolkit for OS/2 Warp (PowerPC Edition).

The chapter includes the following sections:

- [Network Functions](#)
- [Network Data Types](#)

Network Functions

This sections describes the following network functions:

- [NetworkAddConnection](#)
- [NetworkAuthenticateNetwork](#)
- [NetworkAuthenticateResource](#)
- [NetworkAuthenticateServer](#)
- [NetworkDelConnection](#)
- [NetworkDetachNetwork](#)
- [NetworkDetachResource](#)
- [NetworkDetachServer](#)
- [NetworkEnumConnection](#)
- [NetworkEnumNetwork](#)
- [NetworkEnumResource](#)
- [NetworkEnumServer](#)
- [NetworkQueryConnection](#)
- [NetworkQueryNetwork](#)
- [NetworkQueryResource](#)
- [NetworkQueryServer](#)
- [NetworkRegisterNetwork](#)

- [NetworkUnregisterNetwork](#)

NetworkAddConnection

NetworkAddConnection - Syntax

NetworkAddConnection connects a local device name to a remote network resource.

```
#include <nwiapi.h>

PSZ      ComputerName;
PSZ      ResourceName;
PSZ      ConnectName;
ULONG    flType;          /* A bit array indicating the type of resource. All other bits are reserved and must be

Resource type bit mask definitions: */
ULONG    ulreturns;

ulreturns = NetworkAddConnection(ComputerName,
                                ResourceName, ConnectName, flType);
```

NetworkAddConnection Parameter - ComputerName

ComputerName (PSZ) - input

The name of the server that has the network resource.

NetworkAddConnection Parameter - ResourceName

ResourceName (PSZ) - input

The name of the network resource.

NetworkAddConnection Parameter - ConnectName

ConnectName (PSZ) - input

A local device name to connect to the remote resource, or the UNC name of a dynamically shared resource, or a UNC name of any shared resource if a deviceless connection is desired. For a discussion of a dynamically shared resource, see the description of *PathName* in the [RESOURCEINFO](#) structure. The connect name string must include the colon delimiter at the end of the string, except in the case of a deviceless connection, in which case a UNC path is passed.

NetworkAddConnection Parameter - flType

flType (ULONG) - input

A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)

The disk drive.

NET_TYPE_QUEUE(0x00000020)

The printer queue.

NET_TYPE_SERIAL(0x00000040)

The serial device.

NetworkAddConnection Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors encountered.

ERROR_ACCESS_DENIED(5)

Access denied.

ERROR_NOT_ENOUGH_MEMORY(8)

Not enough storage available to process this function.

ERROR_BAD_NETPATH(53)

The network path was not found.

ERROR_BAD_DEV_TYPE(66)

This network device type is incorrect.

ERROR_BAD_NET_NAME(67)

This network name cannot be found.

ERROR_ALREADY_ASSIGNED(85)

Duplicate redirection.

ERROR_INVALID_PARAMETER(87)

The specified parameter is invalid.

ERROR_NETWORK_NOT_FOUND(241)

The specified network is invalid.

MSG_ERROR_LOCAL_DRIVE(2405)

The drive is in local use.

NetworkAddConnection - Parameters

ComputerName (PSZ) - input

The name of the server that has the network resource.

ResourceName (PSZ) - input

The name of the network resource.

ConnectName (PSZ) - input

A local device name to connect to the remote resource, or the UNC name of a dynamically shared resource, or a UNC name of any shared resource if a deviceless connection is desired. For a discussion of a dynamically shared resource, see the description of *PathName* in the [RESOURCEINFO](#) structure. The connect name string must include the colon delimiter at the end of the string, except in the case of a deviceless connection, in which case a UNC path is passed.

fiType (ULONG) - input

A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)

The disk drive.

NET_TYPE_QUEUE(0x00000020)

The printer queue.

NET_TYPE_SERIAL(0x00000040)

The serial device.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors encountered.

ERROR_ACCESS_DENIED(5)

Access denied.

ERROR_NOT_ENOUGH_MEMORY(8)

Not enough storage available to process this function.

ERROR_BAD_NETPATH(53)

The network path was not found.

ERROR_BAD_DEV_TYPE(66)

This network device type is incorrect.

ERROR_BAD_NET_NAME(67)

This network name cannot be found.

ERROR_ALREADY_ASSIGNED(85)

Duplicate redirection.

ERROR_INVALID_PARAMETER(87)

The specified parameter is invalid.

ERROR_NETWORK_NOT_FOUND(241)

The specified network is invalid.

MSG_ERROR_LOCAL_DRIVE(2405)

The drive is in local use.

NetworkAddConnection - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

NetworkAuthenticateNetwork

NetworkAuthenticateNetwork - Syntax

NetworkAuthenticateNetwork authenticates a user for access to a network.

```
#define INCL_xxx
#include <os2.h>
```

```
PSZ      NetworkName;
PSZ      Authenticator;
```

```
PSZ      UserName;  
PSZ      Password;  
ULONG    ulreturns;
```

```
ulreturns = NetworkAuthenticateNetwork(NetworkName,  
                                         Authenticator, UserName, Password);
```

NetworkAuthenticateNetwork Parameter - NetworkName

NetworkName (PSZ) - input
The name of the network.

NetworkAuthenticateNetwork Parameter - Authenticator

Authenticator (PSZ) - input
The name by which networks can route authentication requests; that is, the name of the server or domain that authenticates the user to the network. For example, for some networks, this field might be the logon server; for LAN Server, this field will be the user's logon domain.

NetworkAuthenticateNetwork Parameter - UserName

UserName (PSZ) - input
The name of the user.

NetworkAuthenticateNetwork Parameter - Password

Password (PSZ) - input
The password entered by the user.

NetworkAuthenticateNetwork Return Value - ulreturns

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

```
EXIT_SUCCESS(0)  
    No errors were encountered.  
ERROR_ACCESS_DENIED(5)  
    Access denied.  
ERROR_NOT_ENOUGH_MEMORY(8)
```

Not enough storage available to process this function.
 ERROR_NOT_SUPPORTED(50)
 This request is not supported by the network.
 ERROR_INVALID_PARAMETER(87)
 The specified parameter is invalid.
 ERROR_NETWORK_NOT_FOUND(241)
 Additional data is available, but the buffer is too small.
 MSG_ERROR_AUTHENT_NOT_FOUND(2215)
 The authenticating server or domain is invalid.
 MSG_ERROR_PASSWORD_EXPIRED(2242)
 The password of this user is expired.

NetworkAuthenticateNetwork - Parameters

NetworkName (PSZ) - input
 The name of the network.

Authenticator (PSZ) - input
 The name by which networks can route authentication requests; that is, the name of the server or domain that authenticates the user to the network. For example, for some networks, this field might be the logon server; for LAN Server, this field will be the user's logon domain.

UserName (PSZ) - input
 The name of the user.

Password (PSZ) - input
 The password entered by the user.

ulreturns (ULONG) - returns
 A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
 No errors were encountered.
 ERROR_ACCESS_DENIED(5)
 Access denied.
 ERROR_NOT_ENOUGH_MEMORY(8)
 Not enough storage available to process this function.
 ERROR_NOT_SUPPORTED(50)
 This request is not supported by the network.
 ERROR_INVALID_PARAMETER(87)
 The specified parameter is invalid.
 ERROR_NETWORK_NOT_FOUND(241)
 Additional data is available, but the buffer is too small.
 MSG_ERROR_AUTHENT_NOT_FOUND(2215)
 The authenticating server or domain is invalid.
 MSG_ERROR_PASSWORD_EXPIRED(2242)
 The password of this user is expired.

NetworkAuthenticateNetwork - Remarks

This call must be made by the application when the NETWORKINFO structure indicates the user is not authenticated and that authentication is required. If the password must be entered by the user, it should not be echoed back. After the user is authenticated, the application should zero the password in memory.

NetworkAuthenticateNetwork - Topics

Select an item:

NetworkAuthenticateResource

NetworkAuthenticateResource - Syntax

NetworkAuthenticateResource authenticates a user for access to a resource.

```
#include <nwiapi.h>

PSZ      ComputerName;
PSZ      ResourceName;
ULONG    flType;      /* A bit array indicating the type of resource. All other bits are reserved and must be

Resource type bit mask definitions: */
PSZ      UserName;
PSZ      Password;
ULONG    ulreturns;

ulreturns = NetworkAuthenticateResource(ComputerName,
                                         ResourceName, flType, UserName,
                                         Password);
```

NetworkAuthenticateResource Parameter - ComputerName

ComputerName (PSZ) - input
The name of the server.

NetworkAuthenticateResource Parameter - ResourceName

ResourceName (PSZ) - input
The name of a resource.

NetworkAuthenticateResource Parameter - flType

flType (ULONG) - input
A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)
The disk drive.
NET_TYPE_QUEUE(0x00000020)
The printer queue.
NET_TYPE_SERIAL(0x00000040)
The serial device.

NetworkAuthenticateResource Parameter - UserName

UserName (PSZ) - input
The name of the user.

NetworkAuthenticateResource Parameter - Password

Password (PSZ) - input
The password entered by the user.

NetworkAuthenticateResource Return Value - ulreturns

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_BAD_NET_NAME(67)
The network name cannot be found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkAuthenticateResource - Parameters

ComputerName (PSZ) - input
The name of the server.

ResourceName (PSZ) - input
The name of a resource.

flType (ULONG) - input
A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)
The disk drive.
NET_TYPE_QUEUE(0x00000020)
The printer queue.
NET_TYPE_SERIAL(0x00000040)
The serial device.

UserName (PSZ) - input
The name of the user.

Password (PSZ) - input
The password entered by the user.

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_BAD_NET_NAME(67)
The network name cannot be found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkAuthenticateResource - Remarks

This call must be made by the application when the RESOURCEINFO structure indicates the user is not authenticated and that authentication is required. If the password must be entered by the user, it should not be echoed back. After the user is authenticated, the application should zero the password in memory.

NetworkAuthenticateResource - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

NetworkAuthenticateServer

NetworkAuthenticateServer - Syntax

NetworkAuthenticateServer authenticates a user for access to a server.

```
#include <nwiapi.h>

PSZ      ComputerName;
PSZ      UserName;
PSZ      Password;
ULONG    ulreturns;

ulreturns = NetworkAuthenticateServer(ComputerName,
                                     UserName, Password);
```

NetworkAuthenticateServer Parameter - ComputerName

ComputerName (PSZ) - input
The name of the server.

NetworkAuthenticateServer Parameter - UserName

UserName (PSZ) - input
The name of the user.

NetworkAuthenticateServer Parameter - Password

Password (PSZ) - input
The password entered by the user.

NetworkAuthenticateServer Return Value - ulreturns

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)	No errors were encountered.
ERROR_ACCESS_DENIED(5)	Access denied.

ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkAuthenticateServer - Parameters

ComputerName (PSZ) - input
The name of the server.

UserName (PSZ) - input
The name of the user.

Password (PSZ) - input
The password entered by the user.

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkAuthenticateServer - Remarks

This call must be made by the application when the SERVERINFO structure indicates the user is not authenticated and that authentication is required. If the password must be entered by the user, it should not be echoed back. After the user is authenticated, the application should zero the password in memory.

NetworkAuthenticateServer - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

NetworkDelConnection

NetworkDelConnection - Syntax

NetworkDelConnection disconnects a local device name from a remote network resource.

```
#include <nwiapi.h>

PSZ      ComputerName;
PVOID     Reserved;
PSZ      ConnectName;
ULONG     flType;
ULONG     ulreturns;

ulreturns = NetworkDelConnection(ComputerName,
                                Reserved, ConnectName, flType);
```

NetworkDelConnection Parameter - ComputerName

ComputerName (PSZ) - input
The name of the server that has the network resource.

NetworkDelConnection Parameter - Reserved

Reserved (PVOID) - input
Reserved for future use and must be 0.

NetworkDelConnection Parameter - ConnectName

ConnectName (PSZ) - input
A local device name connected to the remote resource, or the UNC name of a deviceless connection. The connect name string must include the colon delimiter at the end of the string, except in the case of a deviceless connection, in which case a UNC path is passed.

NetworkDelConnection Parameter - flType

flType (ULONG) - input

A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)

The disk drive.

NET_TYPE_QUEUE(0x00000020)

The printer queue.

NET_TYPE_SERIAL(0x00000040)

The serial device.

NetworkDelConnection Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.

ERROR_NOT_ENOUGH_MEMORY(8)

Not enough storage available to process this function.

ERROR_INVALID_DRIVE(15)

The specified drive is not valid.

ERROR_BAD_NETPATH(53)

The network path was not found.

ERROR_BAD_DEV_TYPE(66)

This network device type is incorrect.

ERROR_BAD_NET_NAME(67)

This network name cannot be found.

ERROR_INVALID_PARAMETER(87)

The specified parameter is invalid.

ERROR_NETWORK_NOT_FOUND(241)

The specified network is invalid.

MSG_ERROR_OPEN_FILES(2401)

There are open files on the connection.

NetworkDelConnection - Parameters

ComputerName (PSZ) - input

The name of the server that has the network resource.

Reserved (PVOID) - input

Reserved for future use and must be 0.

ConnectName (PSZ) - input

A local device name connected to the remote resource, or the UNC name of a deviceless connection. The connect name string must include the colon delimiter at the end of the string, except in the case of a deviceless connection, in which case a UNC path is passed.

fiType (ULONG) - input

A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)

The disk drive.

NET_TYPE_QUEUE(0x00000020)

The printer queue.

NET_TYPE_SERIAL(0x00000040)

The serial device.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error

numbers can be returned for errors not listed.

```
EXIT_SUCCESS(0)
    No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
    Not enough storage available to process this function.
ERROR_INVALID_DRIVE(15)
    The specified drive is not valid.
ERROR_BAD_NETPATH(53)
    The network path was not found.
ERROR_BAD_DEV_TYPE(66)
    This network device type is incorrect.
ERROR_BAD_NET_NAME(67)
    This network name cannot be found.
ERROR_INVALID_PARAMETER(87)
    The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
    The specified network is invalid.
MSG_ERROR_OPEN_FILES(2401)
    There are open files on the connection.
```

NetworkDelConnection - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

NetworkDetachNetwork

NetworkDetachNetwork - Syntax

NetworkDetachNetwork detaches the user from access to a network.

```
#include <nwiapi.h>

PSZ      NetworkName;
PSZ      Authenticator;
ULONG    ulreturns;

ulreturns = NetworkDetachNetwork(NetworkName,
    Authenticator);
```

NetworkDetachNetwork Parameter - NetworkName

NetworkName (PSZ) - input
The name of the network.

NetworkDetachNetwork Parameter - Authenticator

Authenticator (PSZ) - input

The name of the server that authenticated the user to the network.

NetworkDetachNetwork Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkDetachNetwork - Parameters

NetworkName (PSZ) - input

The name of the network.

Authenticator (PSZ) - input

The name of the server that authenticated the user to the network.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkDetachNetwork - Topics

Select an item:

[Syntax](#)

NetworkDetachResource

NetworkDetachResource - Syntax

NetworkDetachResource detaches a user from access to a resource.

```
#include <nwiapi.h>
```

```
PSZ      ComputerName;  
PSZ      ResourceName;  
ULONG    flType;  
ULONG    ulreturns;    /*
```

A network driver is not limited to just these errors. Network drivers will return the following errors when they

```
ulreturns = NetworkDetachResource(ComputerName,  
                                   ResourceName, flType);
```

NetworkDetachResource Parameter - ComputerName

ComputerName (PSZ) - input
The name of the server.

NetworkDetachResource Parameter - ResourceName

ResourceName (PSZ) - input
The name of a resource.

NetworkDetachResource Parameter - flType

flType (ULONG) - input
A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

```
NET_TYPE_DISK(0x00000010)  
The disk drive.
```

NET_TYPE_QUEUE(0x00000020)
The printer queue.
NET_TYPE_SERIAL(0x00000040)
The serial device.

NetworkDetachResource Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_BAD_NET_NAME(67)
The network name cannot be found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkDetachResource - Parameters

ComputerName (PSZ) - input
The name of the server.

ResourceName (PSZ) - input
The name of a resource.

fiType (ULONG) - input
A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)
The disk drive.
NET_TYPE_QUEUE(0x00000020)
The printer queue.
NET_TYPE_SERIAL(0x00000040)
The serial device.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_NOT_SUPPORTED(50)
This request is not supported by the network.
ERROR_BAD_NETPATH(53)
The network path was not found.

ERROR_BAD_NET_NAME(67)
The network name cannot be found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkDetachResource - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

NetworkDetachServer

NetworkDetachServer - Syntax

NetworkDetachServer detaches a user from a server.

```
#include <nwiapi.h>

PSZ      ComputerName;
ULONG    ulreturns;

ulreturns = NetworkDetachServer(ComputerName);
```

NetworkDetachServer Parameter - ComputerName

ComputerName (PSZ) - input
The name of the server.

NetworkDetachServer Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.

ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkDetachServer - Parameters

ComputerName (PSZ) - input
The name of the server.

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkDetachServer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

NetworkEnumConnection

NetworkEnumConnection - Syntax

NetworkEnumConnection enumerates connections the user of the workstation has made.

```
#include <nwiapi.h>
```

```
PSZ      NetworkName;  
ULONG    flType;  
ULONG    ulLevel;  
PVOID    pBuf;  
ULONG    cbBuf;
```



```

PULONG    pcReturned;
PULONG    pcTotal;
PULONG    pcbNeeded;
PVOID     pReserved;
ULONG     ulreturns;    /*

```

A network driver is not limited to just these errors. Network drivers will return the following errors when they

```

ulreturns = NetworkEnumConnection(NetworkName,
                                   flType, ulLevel, pBuf, cbBuf,
                                   pcReturned, pcTotal, pcbNeeded,
                                   pReserved);

```

NetworkEnumConnection Parameter - NetworkName

NetworkName (PSZ) - input

The name of the network whose connections are enumerated. NULL enumerates connections on all networks.

NetworkEnumConnection Parameter - flType

flType (ULONG) - input

A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

```

NET_TYPE_DISK(0x00000010)
    The disk drive.
NET_TYPE_QUEUE(0x00000020)
    The printer queue.
NET_TYPE_SERIAL(0x00000040)
    The serial device.

```

NetworkEnumConnection Parameter - ulLevel

ulLevel (ULONG) - input

The level of detail required. This must be 1.

NetworkEnumConnection Parameter - pBuf

pBuf (PVOID) - in/out

The buffer that contains the [CONNECTINFO1](#) data structures:

Input	Used to pass in a pointer to the buffer that will contain the CONNECTINFO1 data structures.
Output	Returns the buffer that contains the CONNECTINFO1 data structures.

NetworkEnumConnection Parameter - cbBuf

cbBuf (ULONG) - input
The size, in bytes, of the buffer.

NetworkEnumConnection Parameter - pcReturned

pcReturned (PULONG) - output
The number of entries returned.

NetworkEnumConnection Parameter - pcTotal

pcTotal (PULONG) - output
The number of entries available.

NetworkEnumConnection Parameter - pcbNeeded

pcbNeeded (PULONG) - output
The size, in bytes, of available information.

NetworkEnumConnection Parameter - pReserved

pReserved (PVOID) - output
Reserved for future use.

NetworkEnumConnection Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.
 ERROR_NOT_ENOUGH_MEMORY(8)
 Not enough storage available to process this function.
 ERROR_INVALID_PARAMETER(87)
 The specified parameter is invalid.
 ERROR_INVALID_LEVEL(124)
 The level parameter is invalid.
 ERROR_MORE_DATA(234)
 Additional data is available, but the buffer is too small.
 ERROR_NETWORK_NOT_FOUND(241)
 The specified network is invalid.

NetworkEnumConnection - Parameters

NetworkName (PSZ) - input

The name of the network whose connections are enumerated. NULL enumerates connections on all networks.

fiType (ULONG) - input

A bit array indicating the type of resource. All other bits are reserved and must be 0.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)

The disk drive.

NET_TYPE_QUEUE(0x00000020)

The printer queue.

NET_TYPE_SERIAL(0x00000040)

The serial device.

ulLevel (ULONG) - input

The level of detail required. This must be 1.

pBuf (PVOID) - in/out

The buffer that contains the [CONNECTINFO1](#) data structures:

Input

Used to pass in a pointer to the buffer that will contain the [CONNECTINFO1](#) data structures.

Output

Returns the buffer that contains the [CONNECTINFO1](#) data structures.

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

pcReturned (PULONG) - output

The number of entries returned.

pcTotal (PULONG) - output

The number of entries available.

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

pReserved (PVOID) - output

Reserved for future use.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.

ERROR_NOT_ENOUGH_MEMORY(8)

Not enough storage available to process this function.

ERROR_INVALID_PARAMETER(87)

The specified parameter is invalid.

ERROR_INVALID_LEVEL(124)

The level parameter is invalid.

ERROR_MORE_DATA(234)

Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkEnumConnection - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

NetworkEnumNetwork

NetworkEnumNetwork - Syntax

NetworkEnumNetwork enumerates networks accessible to the workstation.

```
#include <nwiapi.h>

ULONG      ulLevel;
PVOID      pBuf;
ULONG      cbBuf;
PULONG     pcReturned;
PULONG     pcTotal;
PULONG     pcbNeeded;
PVOID      pReserved;
ULONG      ulreturns;

ulreturns = NetworkEnumNetwork(ulLevel, pBuf,
                               cbBuf, pcReturned, pcTotal,
                               pcbNeeded, pReserved);
```

NetworkEnumNetwork Parameter - ulLevel

ulLevel (ULONG) - input
The level of detail required. This must be 1.

NetworkEnumNetwork Parameter - pBuf

pBuf (PVOID) - in/out
The buffer that contains the [NETWORKINFO1](#) data structures.

NetworkEnumNetwork Parameter - cbBuf

cbBuf (ULONG) - input
The size, in bytes, of the buffer.

NetworkEnumNetwork Parameter - pcReturned

pcReturned (PULONG) - output
The number of entries returned.

NetworkEnumNetwork Parameter - pcTotal

pcTotal (PULONG) - output
The number of entries available.

NetworkEnumNetwork Parameter - pcbNeeded

pcbNeeded (PULONG) - output
The size, in bytes, of available information.

NetworkEnumNetwork Parameter - pReserved

pReserved (PVOID) - output
Reserved for future use.

NetworkEnumNetwork Return Value - ulreturns

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.

NetworkEnumNetwork - Parameters

ulLevel (ULONG) - input

The level of detail required. This must be 1.

pBuf (PVOID) - in/out

The buffer that contains the [NETWORKINFO1](#) data structures.

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

pcReturned (PULONG) - output

The number of entries returned.

pcTotal (PULONG) - output

The number of entries available.

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

pReserved (PVOID) - output

Reserved for future use.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.

NetworkEnumNetwork - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

NetworkEnumResource

NetworkEnumResource - Syntax

NetworkEnumResource enumerates server resources which are accessible to the workstation.

```
#include <nwiapi.h>
```

```
PSZ      ComputerName;  
ULONG    flType;  
ULONG    ulLevel;  
PVOID     pBuf;  
ULONG     cbBuf;  
PULONG    pcReturned;  
PULONG    pcTotal;  
PULONG    pcbNeeded;  
PVOID     pReserved1;  
PVOID     pReserved2;  
ULONG     ulreturns;
```

```
ulreturns = NetworkEnumResource(ComputerName,  
                                flType, ulLevel, pBuf, cbBuf,  
                                pcReturned, pcTotal, pcbNeeded,  
                                pReserved1, pReserved2);
```

NetworkEnumResource Parameter - ComputerName

ComputerName (PSZ) - input

The name of the server whose resources are enumerated.

NetworkEnumResource Parameter - flType

flType (ULONG) - input

A bit array indicating the type of resources to enumerate. All other server type bits are reserved and must be 0.

Resource type bit mask definitions:

```
NET_AUTHENTICATED(0x00000001)  
    Enumerate authenticated resources only.  
NET_TYPE_DISK(0x00000010)  
    The disk drive.  
NET_TYPE_QUEUE(0x00000020)  
    The printer queue.  
NET_TYPE_SERIAL(0x00000040)  
    The serial device.
```

NetworkEnumResource Parameter - ulLevel

ulLevel (ULONG) - input

The level of detail required. This must be 1 or 2.

NetworkEnumResource Parameter - pBuf

pBuf (PVOID) - in/out

If level 1, the buffer that contains the [RESOURCEINFO1](#) data structures. If level 2, the buffer that contains the [RESOURCEINFO2](#) data structures.

NetworkEnumResource Parameter - cbBuf

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

NetworkEnumResource Parameter - pcReturned

pcReturned (PULONG) - output

The number of entries returned.

NetworkEnumResource Parameter - pcTotal

pcTotal (PULONG) - output

The number of entries available.

NetworkEnumResource Parameter - pcbNeeded

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

NetworkEnumResource Parameter - pReserved1

pReserved1 (PVOID) - output

Reserved for future use and must be 0.

NetworkEnumResource Parameter - pReserved2

pReserved2 (PVOID) - output
Reserved for future use.

NetworkEnumResource Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkEnumResource - Parameters

ComputerName (PSZ) - input
The name of the server whose resources are enumerated.

fiType (ULONG) - input
A bit array indicating the type of resources to enumerate. All other server type bits are reserved and must be 0.

Resource type bit mask definitions:

NET_AUTHENTICATED(0x00000001)
Enumerate authenticated resources only.
NET_TYPE_DISK(0x00000010)
The disk drive.
NET_TYPE_QUEUE(0x00000020)
The printer queue.
NET_TYPE_SERIAL(0x00000040)
The serial device.

ulLevel (ULONG) - input
The level of detail required. This must be 1 or 2.

pBuf (PVOID) - in/out
If level 1, the buffer that contains the [RESOURCEINFO1](#) data structures. If level 2, the buffer that contains the [RESOURCEINFO2](#) data structures.

cbBuf (ULONG) - input
The size, in bytes, of the buffer.

pcReturned (PULONG) - output
The number of entries returned.

pcTotal (PULONG) - output
The number of entries available.

pcbNeeded (PULONG) - output
The size, in bytes, of available information.

pReserved1 (PVOID) - output
Reserved for future use and must be 0.

pReserved2 (PVOID) - output
Reserved for future use.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.

ERROR_ACCESS_DENIED(5)
Access denied.

ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.

ERROR_BAD_NETPATH(53)
The network path was not found.

ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.

ERROR_INVALID_LEVEL(124)
The level parameter is invalid.

ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.

ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkEnumResource - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

NetworkEnumServer

NetworkEnumServer - Syntax

NetworkEnumServer enumerates servers.

```
#include <nwiapi.h>
```

```
PSZ      NetworkName;
ULONG    flType;
ULONG    ulLevel;
PVOID     pBuf;
ULONG     cbBuf;
PULONG    pcReturned;
PULONG    pcTotal;
PULONG    pcbNeeded;
PVOID     pReserved1;
PVOID     pReserved2;
ULONG     ulreturns;
```

```
ulreturns = NetworkEnumServer(NetworkName,
                                flType, ulLevel, pBuf, cbBuf,
                                pcReturned, pcTotal, pcbNeeded,
                                pReserved1, pReserved2);
```

NetworkEnumServer Parameter - NetworkName

NetworkName (PSZ) - input
The name of the network.

NetworkEnumServer Parameter - flType

flType (ULONG) - input
A bit array indicating the type of servers to enumerate. All other server-type bits are reserved and must be 0.

Server type bit definitions:

NET_AUTHENTICATED(0x00000001)
Enumerate authenticated servers only.

NetworkEnumServer Parameter - ulLevel

ulLevel (ULONG) - input
The level of detail required. This must be 1 or 2.

NetworkEnumServer Parameter - pBuf

pBuf (PVOID) - in/out
The address of the [SERVERINFO1](#) or [SERVERINFO2](#) data structure (depending on the level specified-1 or 2).

NetworkEnumServer Parameter - cbBuf

cbBuf (ULONG) - input
The size, in bytes, of the buffer.

NetworkEnumServer Parameter - pcReturned

pcReturned (PULONG) - output
The number of entries returned.

NetworkEnumServer Parameter - pcTotal

pcTotal (PULONG) - output
The number of entries available.

NetworkEnumServer Parameter - pcbNeeded

pcbNeeded (PULONG) - output
The size, in bytes, of available information.

NetworkEnumServer Parameter - pReserved1

pReserved1 (PVOID) - output
Reserved for future use and must be 0.

NetworkEnumServer Parameter - pReserved2

pReserved2 (PVOID) - output
Reserved for future use.

NetworkEnumServer Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkEnumServer - Parameters

NetworkName (PSZ) - input

The name of the network.

fiType (ULONG) - input

A bit array indicating the type of servers to enumerate. All other server-type bits are reserved and must be 0.

Server type bit definitions:

NET_AUTHENTICATED(0x00000001)
Enumerate authenticated servers only.

ulLevel (ULONG) - input

The level of detail required. This must be 1 or 2.

pBuf (PVOID) - in/out

The address of the [SERVERINFO1](#) or [SERVERINFO2](#) data structure (depending on the level specified-1 or 2).

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

pcReturned (PULONG) - output

The number of entries returned.

pcTotal (PULONG) - output

The number of entries available.

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

pReserved1 (PVOID) - output

Reserved for future use and must be 0.

pReserved2 (PVOID) - output

Reserved for future use.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)

```
        The level parameter is invalid.
ERROR_MORE_DATA(234)
        Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
        The specified network is invalid.
```

NetworkEnumServer - Remarks

Some servers might be enumerated that are no longer accessible because of a delay between the time a server is removed and the time it becomes inaccessible from the enumeration list. To verify the most recent access information, an application should query the server with NetworkQueryServer.

NetworkEnumServer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

NetworkQueryConnection

NetworkQueryConnection - Syntax

NetworkQueryConnection queries what remote network resource a local device name is connected to.

```
#include <nwiapi.h>

PSZ          ConnectName;
ULONG        ulLevel;
PVOID        pBuf;
ULONG        cbBuf;
PULONG       pcbNeeded;
ULONG        ulreturns;

ulreturns = NetworkQueryConnection(ConnectName,
                                   ulLevel, pBuf, cbBuf, pcbNeeded);
```

NetworkQueryConnection Parameter - ConnectName

ConnectName (PSZ) - input

The name of the local device that is connected to a remote resource, or the UNC name of a shared resource with a deviceless connection. The connect name string must include the colon delimiter at the end of the string, except in the case of a deviceless

connection, in which case a UNC path is passed.

NetworkQueryConnection Parameter - ulLevel

ulLevel (ULONG) - input
The level of detail required. This must be 1.

NetworkQueryConnection Parameter - pBuf

pBuf (PVOID)
The buffer that contains the [CONNECTINFO1](#) data structures.

NetworkQueryConnection Parameter - cbBuf

cbBuf (ULONG) - input
The size, in bytes, of the buffer.

NetworkQueryConnection Parameter - pcbNeeded

pcbNeeded (PULONG) - output
The size, in bytes, of available information.

NetworkQueryConnection Return Value - ulreturns

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NERR_UseNotFound(2250)
The connection can not be found.

NetworkQueryConnection - Parameters

ConnectName (PSZ) - input

The name of the local device that is connected to a remote resource, or the UNC name of a shared resource with a deviceless connection. The connect name string must include the colon delimiter at the end of the string, except in the case of a deviceless connection, in which case a UNC path is passed.

ulLevel (ULONG) - input

The level of detail required. This must be 1.

pBuf (PVOID)

The buffer that contains the [CONNECTINFO1](#) data structures.

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.

ERROR_NOT_ENOUGH_MEMORY(8)

Not enough storage available to process this function.

ERROR_INVALID_PARAMETER(87)

The specified parameter is invalid.

ERROR_INVALID_LEVEL(124)

The level parameter is invalid.

ERROR_MORE_DATA(234)

Additional data is available, but the buffer is too small.

ERROR_NETWORK_NOT_FOUND(241)

The specified network is invalid.

NERR_UseNotFound(2250)

The connection can not be found.

NetworkQueryConnection - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

NetworkQueryNetwork

NetworkQueryNetwork - Syntax

NetworkQueryNetwork queries a network.

```
#include <nwiapi.h>
```

```
PSZ      NetworkName;  
ULONG    ulLevel;  
PVOID    pBuf;  
ULONG    cbBuf;  
PULONG   pcbNeeded;  
ULONG    ulreturns;
```

```
ulreturns = NetworkQueryNetwork(NetworkName,  
                                ulLevel, pBuf, cbBuf, pcbNeeded);
```

NetworkQueryNetwork Parameter - NetworkName

NetworkName (PSZ) - input
The name of the network.

NetworkQueryNetwork Parameter - ulLevel

ulLevel (ULONG) - input
The level of detail required. This must be 1.

NetworkQueryNetwork Parameter - pBuf

pBuf (PVOID)
The buffer that contains the [NETWORKINFO1](#) data structures.

NetworkQueryNetwork Parameter - cbBuf

cbBuf (ULONG) - input
The size, in bytes, of the buffer.

NetworkQueryNetwork Parameter - pcbNeeded

pcbNeeded (PULONG) - output
The size, in bytes, of available information.

NetworkQueryNetwork Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the above errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkQueryNetwork - Parameters

NetworkName (PSZ) - input

The name of the network.

ulLevel (ULONG) - input

The level of detail required. This must be 1.

pBuf (PVOID)

The buffer that contains the [NETWORKINFO1](#) data structures.

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the above errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkQueryNetwork - Topics

Select an item:

NetworkQueryResource

NetworkQueryResource - Syntax

NetworkQueryResource queries a resource.

```
#include <nwiapi.h>

PSZ      ComputerName;
PSZ      ResourceName;
ULONG    flType;
ULONG    ulLevel;
PVOID    pBuf;
ULONG    cbBuf;
PULONG   pcbNeeded;
ULONG    ulreturns;

ulreturns = NetworkQueryResource(ComputerName,
                                ResourceName, flType, ulLevel,
                                pBuf, cbBuf, pcbNeeded);
```

NetworkQueryResource Parameter - ComputerName

ComputerName (PSZ) - input
The name of the server to be queried.

NetworkQueryResource Parameter - ResourceName

ResourceName (PSZ) - input
The name of the resource to be queried.

NetworkQueryResource Parameter - flType

flType (ULONG) - input
A bit array indicating the type of resource.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)

The disk drive.

NET_TYPE_QUEUE(0x00000020)

The printer queue.

NET_TYPE_SERIAL(0x00000040)

The serial device.

NetworkQueryResource Parameter - ulLevel

ulLevel (ULONG) - input

The level of detail required. This must be 1 or 2.

NetworkQueryResource Parameter - pBuf

pBuf (PVOID) - in/out

The buffer that contains the [RESOURCEINFO1](#) or [RESOURCEINFO2](#) data structure (depending on the level).

NetworkQueryResource Parameter - cbBuf

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

NetworkQueryResource Parameter - pcbNeeded

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

NetworkQueryResource Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when then occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.

ERROR_ACCESS_DENIED(5)

Access denied.

ERROR_NOT_ENOUGH_MEMORY(8)

Not enough storage available to process this function.
ERROR_BAD_NET_PATH(53)
The network path was not found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkQueryResource - Parameters

ComputerName (PSZ) - input
The name of the server to be queried.

ResourceName (PSZ) - input
The name of the resource to be queried.

flType (ULONG) - input
A bit array indicating the type of resource.

Resource type bit mask definitions:

NET_TYPE_DISK(0x00000010)
The disk drive.
NET_TYPE_QUEUE(0x00000020)
The printer queue.
NET_TYPE_SERIAL(0x00000040)
The serial device.

ulLevel (ULONG) - input
The level of detail required. This must be 1 or 2.

pBuf (PVOID) - in/out
The buffer that contains the [RESOURCEINFO1](#) or [RESOURCEINFO2](#) data structure (depending on the level).

cbBuf (ULONG) - input
The size, in bytes, of the buffer.

pcbNeeded (PULONG) - output
The size, in bytes, of available information.

ulreturns (ULONG) - returns
A network driver is not limited to just these errors. Network drivers will return the following errors when then occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_BAD_NET_PATH(53)
The network path was not found.
ERROR_INVALID_PARAMETER(87)
The specified parameter is invalid.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkQueryResource - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

NetworkQueryServer

NetworkQueryServer - Syntax

NetworkQueryServer queries a server.

```
#include <nwiapi.h>
```

```
PSZ      ComputerName;  
ULONG    flType;  
ULONG    ulLevel;  
PVOID    pBuf;  
ULONG    cbBuf;  
PULONG   pcbNeeded;  
ULONG    ulreturns;
```

```
ulreturns = NetworkQueryServer(ComputerName,  
                                flType, ulLevel, pBuf, cbBuf,  
                                pcbNeeded);
```

NetworkQueryServer Parameter - ComputerName

ComputerName (PSZ) - input

The name of the server to be queried.

NetworkQueryServer Parameter - flType

flType (ULONG) - input

The type of the servers to enumerate. Currently reserved and must be 0.

NetworkQueryServer Parameter - ulLevel

ulLevel (ULONG) - input

The level of detail required. This must be 1 or 2.

NetworkQueryServer Parameter - pBuf

pBuf (PVOID) - in/out

The buffer that contains the [SERVERINFO1](#) or [SERVERINFO2](#) data structures.

NetworkQueryServer Parameter - cbBuf

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

NetworkQueryServer Parameter - pcbNeeded

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

NetworkQueryServer Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_ACCESS_DENIED(5)
Access denied.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_BAD_NETPATH(53)
The network path was not found.
ERROR_BAD_NET_NAME(67)
The network name cannot be found.
ERROR_INVALID_LEVEL(124)
The level parameter is invalid.
ERROR_MORE_DATA(234)
Additional data is available, but the buffer is too small.
ERROR_NETWORK_NOT_FOUND(241)
The specified network is invalid.

NetworkQueryServer - Parameters

ComputerName (PSZ) - input

The name of the server to be queried.

fiType (ULONG) - input

The type of the servers to enumerate. Currently reserved and must be 0.

ulLevel (ULONG) - input

The level of detail required. This must be 1 or 2.

pBuf (PVOID) - in/out

The buffer that contains the [SERVERINFO1](#) or [SERVERINFO2](#) data structures.

cbBuf (ULONG) - input

The size, in bytes, of the buffer.

pcbNeeded (PULONG) - output

The size, in bytes, of available information.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.

ERROR_ACCESS_DENIED(5)

Access denied.

ERROR_NOT_ENOUGH_MEMORY(8)

Not enough storage available to process this function.

ERROR_BAD_NETPATH(53)

The network path was not found.

ERROR_BAD_NET_NAME(67)

The network name cannot be found.

ERROR_INVALID_LEVEL(124)

The level parameter is invalid.

ERROR_MORE_DATA(234)

Additional data is available, but the buffer is too small.

ERROR_NETWORK_NOT_FOUND(241)

The specified network is invalid.

NetworkQueryServer - Remarks

The server is verified to be accessible on the network when this function returns without an error.

NetworkQueryServer - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

NetworkRegisterNetwork

NetworkRegisterNetwork - Syntax

NetworkRegisterNetwork registers a worker DLL with the network-independent router for a specific network.

```
#include <nwiapi.h>

PSZ      DriverName;
PSZ      NetworkName;
ULONG    ulreturns;

ulreturns = NetworkRegisterNetwork(DriverName,
                                   NetworkName);
```

NetworkRegisterNetwork Parameter - DriverName

DriverName (PSZ) - input

The name of the worker DLL; for example NWORKER. The syntax of the name is the same as for DosLoadModule.

NetworkRegisterNetwork Parameter - NetworkName

NetworkName (PSZ) - input

The name of the network; for example NW. It is recommended that two letters be used for the network name; a longer name can be used for the network description.

NetworkRegisterNetwork Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

```
EXIT_SUCCESS(0)
    No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
    Not enough storage available to process this function.
ERROR_DUPLICATE_NETWORK_NAME(242)
    Duplicate network name.
```

NetworkRegisterNetwork - Parameters

DriverName (PSZ) - input

The name of the worker DLL; for example NWORKER. The syntax of the name is the same as for DosLoadModule.

NetworkName (PSZ) - input

The name of the network; for example NW. It is recommended that two letters be used for the network name; a longer name can be used for the network description.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)
No errors were encountered.
ERROR_NOT_ENOUGH_MEMORY(8)
Not enough storage available to process this function.
ERROR_DUPLICATE_NETWORK_NAME(242)
Duplicate network name.

NetworkRegisterNetwork - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

NetworkUnregisterNetwork

NetworkUnregisterNetwork - Syntax

NetworkUnregisterNetwork unregisters a worker DLL's network with the network-independent router.

```
#include <nwiapi.h>

PSZ      NetworkName;
ULONG    ulreturns;

ulreturns = NetworkUnregisterNetwork(NetworkName);
```

NetworkUnregisterNetwork Parameter - NetworkName

NetworkName (PSZ) - input

The name of the network that is to be unregistered; for example, NW.

NetworkUnregisterNetwork Return Value - ulreturns

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.

ERROR_NETWORK_NOT_FOUND(241)

Not enough storage available to process this function.

NetworkUnregisterNetwork - Parameters

NetworkName (PSZ) - input

The name of the network that is to be unregistered; for example, NW.

ulreturns (ULONG) - returns

A network driver is not limited to just these errors. Network drivers will return the following errors when they occur, but other error numbers can be returned for errors not listed.

EXIT_SUCCESS(0)

No errors were encountered.

ERROR_NETWORK_NOT_FOUND(241)

Not enough storage available to process this function.

NetworkUnregisterNetwork - Remarks

NetworkUnregisterNetwork unregisters the network name so that new processes will not route to the worker DLL; processes that have already initialized will still have a valid link to the worker DLL.

NetworkUnregisterNetwork - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

Network Data Types

This section describes the following data types that are used with the Network functions:

- [CONNECTINFO1](#)
- [NETWORKINFO1](#)
- [RESOURCEINFO1](#)
- [RESOURCEINFO2](#)
- [SERVERINFO1](#)
- [SERVERINFO2](#)

CONNECTINFO1

Connection information data structure.

```
typedef struct _CONNECTINFO1 {
    PSZ      ComputerName;
    PSZ      ResourceName;
    PSZ      ConnectName;
    ULONG     flType;          /* A bit array indicating the type of resource.

Resource type bit definitions: */
} CONNECTINFO1;

typedef CONNECTINFO1 *PCONNECTINFO1;
```

CONNECTINFO1 Field - ComputerName

ComputerName (PSZ)

The computername of the server providing the resource.

CONNECTINFO1 Field - ResourceName

ResourceName (PSZ)

The remote network resource name.

CONNECTINFO1 Field - ConnectName

ConnectName (PSZ)

The local device name connected to the network resource, or the UNC name of a shared resource with a deviceless connection. The connect name string includes the colon delimiter at the end of the string, except in the case of a deviceless connection; in this case, the UNC path is passed.

CONNECTINFO1 Field - flType

flType (ULONG)

A bit array indicating the type of resource.

Resource type bit definitions:

NET_TYPE_DISK (0x00000010)

The disk drive.

NET_TYPE_QUEUE (0x00000020)

The printer queue.

NET_TYPE_SERIAL (0x00000040)
The serial device.

NETWORKINFO1

Network information data structure.

```
typedef struct _NETWORKINFO1 {  
    PSZ      NetworkName;  
    USHORT   usVersionMajor;  
    USHORT   usVersionMinor;  
    ULONG     flInfo;                /* Information flag bit definitions: */  
    ULONG     flCapabilities[5];  
    PSZ      DefaultAuthenticator;  
    PSZ      UserName;  
    PSZ      MessageFile;  
    PSZ      Comment;  
} NETWORKINFO1;  
  
typedef NETWORKINFO1 *PNETWORKINFO1;
```

NETWORKINFO1 Field - NetworkName

NetworkName (PSZ)
The name of the network.

NETWORKINFO1 Field - usVersionMajor

usVersionMajor (USHORT)
The major release version number of the network driver running on this workstation.

NETWORKINFO1 Field - usVersionMinor

usVersionMinor (USHORT)
The minor release version number of the network driver running on this workstation.

NETWORKINFO1 Field - flInfo

flInfo (ULONG)
Information flag bit definitions:

NET_AUTHENTICATED(0x00000001)
The user is authenticated.
NET_MUST_AUTHENTICATE(0x00000002)
Authentication is required.

NETWORKINFO1 Field - flCapabilities[5]

flCapabilities[5] (ULONG)

An array of flags indicating the capabilities of the network to support network, job, queue, and device API functions.

Network flCapabilities[0] flag bit definitions:

NET_SUP_AUTHENTICATENETWORK (0x00000001)
NET_SUP_AUTHENTICATESERVER (0x00000002)
NET_SUP_AUTHENTICATERESOURCE (0x00000004)
NET_SUP_DETACHNETWORK (0x00000008)
NET_SUP_DETACHSERVER (0x00000010)
NET_SUP_DETACHRESOURCE (0x00000020)
NET_SUP_ENUMNETWORK (0x00000040)
NET_SUP_QUERYNETWORK (0x00000080)
NET_SUP_ENUMSERVER (0x00000100)
NET_SUP_QUERYSERVER (0x00000200)
NET_SUP_ENUMRESOURCE (0x00000400)
NET_SUP_QUERYRESOURCE (0x00000800)
NET_SUP_ADDCONNECTION (0x00001000)
NET_SUP_DELCONNECTION (0x00002000)
NET_SUP_ENUMCONNECTION (0x00004000)
NET_SUP_QUERYCONNECTION (0x00008000)

flCapabilities[1] flag bit definitions:

NET_SUP_SPLCONTROLDEVICE (0x00000001)
NET_SUP_SPLCREATEDevice (0x00000002)
NET_SUP_SPLDELETEDevice (0x00000004)
NET_SUP_SPLENUMDEVICE (0x00000008)
NET_SUP_SPLQUERYDEVICE (0x00000010)
NET_SUP_SPLSETDEVICE (0x00000020)
NET_SUP_SPLENUMDRIVER (0x00010000)
NET_SUP_SPLENUMPORT (0x00020000)

Job flCapabilities[2] flag bit definitions:

NET_SUP_SPLCOPYJOB (0x00000001)
NET_SUP_SPLDELETEJOB (0x00000002)
NET_SUP_SPLENUMJOB (0x00000004)
NET_SUP_SPLHOLDJOB (0x00000008)
NET_SUP_SPLQUERYJOB (0x00000010)
NET_SUP_SPLRELEASEJOB (0x00000020)
NET_SUP_SPLSETJOB (0x00000040)

NetworkAuthenticateNetwork is supported.
NetworkAuthenticateServer is supported.
NetworkAuthenticateResource is supported.
NetworkDetachNetwork is supported.
NetworkDetachServer is supported.
NetworkDetachResource is supported.
NetworkEnumNetwork is supported.
NetworkQueryNetwork is supported.
NetworkEnumServer is supported.
NetworkQueryServer is supported.
NetworkEnumResource is supported.
NetworkQueryResource is supported.
NetworkAddConnection is supported.
NetworkDelConnection is supported.
NetworkEnumConnection is supported.
NetworkQueryConnection is supported. *Device*

SplControlDevice is supported.
SplCreateDevice is supported.
SplDeleteDevice is supported.
SplEnumDevice is supported.
SplQueryDevice is supported.
SplSetDevice is supported.
SplEnumDriver is supported.
SplEnumPort is supported.

SplCopyJob is supported.
SplDeleteJob is supported.
SplEnumJob is supported.
SplHoldJob is supported.
SplQueryJob is supported.
SplReleaseJob is supported.
SplSetJob is supported.

Queue flCapabilities[3] flag bit definitions:

NET_SUP_SPLCREATEQUEUE (0x00000001)

NET_SUP_SPLDELETEQUEUE (0x00000002)

NET_SUP_SPLENUMQUEUE (0x00000004)

NET_SUP_SPLHOLDQUEUE (0x00000008)

NET_SUP_SPLPURGEQUEUE (0x00000010)

NET_SUP_SPLQUERYQUEUE (0x00000020)

NET_SUP_SPLRELEASEQUEUE (0x00000040)

NET_SUP_SPLSETQUEUE (0x00000080)

NET_SUP_SPLENUMQUEUEPROCESSOR (0x00010000)

SplCreateQueue is supported.
SplDeleteQueue is supported.
SplEnumQueue is supported.
SplHoldQueue is supported.
SplPurgeQueue is supported.
SplQueryQueue is supported.
SplReleaseQueue is supported.
SplSetQueue is supported.
SplEnumQueueProcessor is supported. *Job submission*

flCapabilities[4] flag bit definitions:

NET_SUP_SPLQMABORT (0x00000001)

NET_SUP_SPLQMABORTDOC (0x00000002)

NET_SUP_SPLQMCLOSE (0x00000004)

NET_SUP_SPLQMENDDOC (0x00000008)

NET_SUP_SPLQMOPEN (0x00000010)

NET_SUP_SPLQMSTARTDOC (0x00000020)

NET_SUP_SPLQMWRITE (0x00000040)

SplQmAbort is supported.
SplQmAbortDoc is supported.
SplQmClose is supported.
SplQmEndDoc is supported.
SplQmOpen is supported.
SplQmStartDoc is supported.
SplQmWrite is supported.

NETWORKINFO1 Field - DefaultAuthenticator

DefaultAuthenticator (PSZ)

The name by which networks can route authentication requests. That is, the name of the server or domain that authenticates the user to the network. For example, for some networks, this field might be the logon server; for the LAN Server, this field will be the user's logon domain.

NETWORKINFO1 Field - UserName

UserName (PSZ)

If the user is authenticated, this is the name the user is authenticated with.

If the user must be authenticated, this is the default name to authenticate the user with.

This may be NULL, if the user is not authenticated or if there is no default name to authenticate the user with.

NETWORKINFO1 Field - MessageFile

MessageFile (PSZ)

The name of a message file that contains messages for network- specific error return codes. This may be NULL.

NETWORKINFO1 Field - Comment

Comment (PSZ)

The comment describing the workstation software. This may be NULL.

RESOURCEINFO1

Resource information data structure 1.

```
typedef struct _RESOURCEINFO1 {
    PSZ      ResourceName;
    PSZ      ConnectName;
    ULONG    flInfo;          /* An array of bits that indicate additional information about the resource.

Information flag bit definitions: */
    ULONG    flCapabilities; /* An array of flags indicating the capabilities of the user to the resource. Net
    ULONG    flType;         /* A bit array indicating the type of resource.

Resource type bit definitions: */
    PSZ      UserName;
    PSZ      PathName;
    PSZ      Comment;
} RESOURCEINFO1;

typedef RESOURCEINFO1 *PRESOURCEINFO1;
```

RESOURCEINFO1 Field - ResourceName

ResourceName (PSZ)

The name of the resource.

RESOURCEINFO1 Field - ConnectName

ConnectName (PSZ)

The name of any connection to this resource. This may be NULL. If there are multiple connections to this resource, only one connection name is given. The connect name string includes the colon delimiter at the end of the string unless this is a deviceless connection, in which case the UNC path is passed.

RESOURCEINFO1 Field - flInfo

flInfo (ULONG)

An array of bits that indicate additional information about the resource.

Information flag bit definitions:

NET_AUTHENTICATED(0x00000001)
The user is authenticated.
NET_MUST_AUTHENTICATE(0x00000002)
Authentication is required.
NET_MUST_CONNECT(0x00000004)
Connection is required.

RESOURCEINFO1 Field - flCapabilities

flCapabilities (ULONG)

An array of flags indicating the capabilities of the user to the resource. *Network Capabilities bit definitions:*

NET_JOBS_CREATE(0x00000001)
The user can create jobs.
NET_JOBS_MANAGE_OWN(0x00000002)
The user can control and delete his or her own jobs.
NET_JOBS_MANAGE_ALL(0x00000004)
The user can control and delete others' jobs.

RESOURCEINFO1 Field - flType

flType (ULONG)

A bit array indicating the type of resource.

Resource type bit definitions:

NET_TYPE_DISK(0x00000010)
The disk drive.
NET_TYPE_QUEUE(0x00000020)
The printer queue.
NET_TYPE_SERIAL(0x00000040)
The serial device.

RESOURCEINFO1 Field - UserName

UserName (PSZ)

If the user is authenticated, this is the name the user is authenticated with.

If the user must be authenticated, this is the default name to authenticate the user with.

This may be NULL, if the user is not authenticated or if there is no default name to authenticate the user with.

RESOURCEINFO1 Field - PathName

PathName (PSZ)

A fully qualified extended path name that is an extension of the UNC name with a syntax "network:\\server\\path". The "network:" must

be stripped to use this path name as a UNC path name, where UNC names are accepted (for example, FileName parameter to DosOpen).

In the case of aliased resources, this parameter contains the real extended UNC path to the resource (this is what should be passed to the file system or print API). Again, the "network:" must be stripped to use this path name as a UNC path name.

If NET_MUST_CONNECT is set in flInfo for a dynamically shared resource, the UNC portion of pszPathName is usable only after a connection to the resource has been made. The connection is made by using the UNC substring of pszPathName as the pszConnectName input parameter to the NetworkAddConnection call. When UNC name access to the resource is no longer needed, NetworkDeleteConnection should be called, so the server that is sharing the dynamically shared resource can remove the share from the network and reclaim the resources associated with the share.

RESOURCEINFO1 Field - Comment

Comment (PSZ)

The comment describing the resource. This may be NULL.

RESOURCEINFO2

Resource information data structure 2.

```
typedef struct _RESOURCEINFO2 {
    PSZ      ResourceName;
    ULONG    flType;        /* A bit array indicating the type of resource.  Resource type bit definitions: */
    PSZ      PathName;
    PSZ      Comment;
} RESOURCEINFO2;

typedef RESOURCEINFO2 *PRESOURCEINFO2;
```

RESOURCEINFO2 Field - ResourceName

ResourceName (PSZ)

The name of the resource.

RESOURCEINFO2 Field - flType

flType (ULONG)

A bit array indicating the type of resource. *Resource type bit definitions:*

```
NET_TYPE_DISK(0x00000010)
    The disk drive.
NET_TYPE_QUEUE(0x00000020)
    The printer queue.
NET_TYPE_SERIAL(0x00000040)
    The serial device.
```

RESOURCEINFO2 Field - PathName

PathName (PSZ)

A fully qualified extended path name that is an extension of the UNC name with a syntax "network:\\server\path". Currently, the "network:" must be stripped to use this path name as a UNC path name wherever UNC names are accepted (for example, FileName parameter to DosOpen).

In the case of aliased resources, this parameter contains the real extended UNC path to the resource (this is what should be passed to the file system or print API). Again, the "network:" must be stripped to use this path name as a UNC path name.

If NET_MUST_CONNECT is set in flInfo for a dynamically shared resource, the UNC portion of pszPathName is usable only after a connection to the resource has been made. The connection is made by using the UNC substring of pszPathName as the pszConnectName input parameter to the NetworkAddConnection call. When UNC name access to the resource is no longer needed, NetworkDeleteConnection should be called so the server that is sharing the dynamically shared resource can remove the share from the network and reclaim the resources associated with the share.

RESOURCEINFO2 Field - Comment

Comment (PSZ)

The comment describing the resource. This may be NULL.

SERVERINFO1

Server information data structure.

```
typedef struct _SERVERINFO1 {
    PSZ      ComputerName;
    USHORT   usVersionMajor;
    USHORT   usVersionMinor;
    ULONG    flInfo;          /* An array of bits that indicates additional information about the server.

Information flag bit definitions: */
    PSZ      UserName;
    PSZ      Comment;
} SERVERINFO1;

typedef SERVERINFO1 *PSERVERINFO1;
```

SERVERINFO1 Field - ComputerName

ComputerName (PSZ)

The computer name of the server.

SERVERINFO1 Field - usVersionMajor

usVersionMajor (USHORT)

The major release version number of the network driver running on the server. The version number may be 0, if unknown.

SERVERINFO1 Field - usVersionMinor

usVersionMinor (USHORT)

The minor release version number of the network driver running on the server. The version number may be 0, if unknown.

SERVERINFO1 Field - flInfo

flInfo (ULONG)

An array of bits that indicates additional information about the server.

Information flag bit definitions:

NET_AUTHENTICATED(0x00000001)

The user is authenticated.

NET_MUST_AUTHENTICATE(0x00000002)

Authentication is required.

SERVERINFO1 Field - UserName

UserName (PSZ)

If the user is authenticated, this is the name the user is authenticated with.

If the user must be authenticated, this is the default name to authenticate the user with.

This may be NULL, if the user is not authenticated or if there is no default name to authenticate the user with.

SERVERINFO1 Field - Comment

Comment (PSZ)

Comment describing the server. It may be NULL.

SERVERINFO2

Server information data structure 2.

```
typedef struct _SERVERINFO2 {
    PSZ      ComputerName;
    PSZ      Comment;
} SERVERINFO2;

typedef SERVERINFO2 *PSERVERINFO2;
```

SERVERINFO2 Field - ComputerName

ComputerName (PSZ)

The computer name of the server.

SERVERINFO2 Field - Comment

Comment (PSZ)

Comment describing the server. It may be NULL.

Problem Determination

Problem Determination services provide basic support for identifying and isolating errors. They include an integrated Error Log to keep a historical record of errors detected by the operating system and, optionally, by applications. The mechanism for inserting entries into the Error Log is the [FFSTProbe](#) function. FFST (First Failure Support Technology) enables errors to be logged when first detected.

[FFSTProbe](#) also supports the collection of trace and user-specified information and the associaton of it with an error log entry and possibly an FFST dump. Traces are used to create a historical record of activity in the operating system and, optionally, applications. FFST dumps are produced if the user calls [FFSTProbe](#) with certain parameters filled in.

A system dump is used to create a snapshot of the entire contents of main memory at the time of a system crash. In most cases, a system dump should be taken only with the assistance of a Service Representative.

The information in the Error Log, along with any trace or dump information, can be valuable in isolating errors more quickly and with less disruption to users.

Problem Determination services provide application programming interfaces (APIs) for:

- Managing Error Log files
- Reading and formatting Error Log entries
- Registering and waiting for Error Log event notifications
- Changing Error Log event-notification filters
- Logging an error and storing associated trace and dump information in a file
- Creating a trace point

The following libraries must be linked with object files that use the Problem Determination functions:

Library	Functions
libffst.a	FFST functions
liblfapi.a	Log functions
libtrace.a	TraceCreateEntry

This chapter includes the following sections:

- [Problem Determination Functions](#)
- [Problem Determination Data Types](#)

Problem Determination Functions

This sections describes the following Problem Determination functions:

- [FFSTProbe](#)
- [FFSTQueryConfiguration](#)
- [FFSTSetConfiguration](#)
- [LogChangeEventFilter](#)
- [LogCloseEventNotification](#)
- [LogCloseFile](#)
- [LogFormatEntry](#)
- [LogOpenEventNotification](#)
- [LogOpenFile](#)
- [LogReadEntry](#)
- [LogWaitEvent](#)
- [TraceCreateEntry](#)

FFSTProbe

FFSTProbe - Syntax

FFSTProbe performs the requested services and returns control to the caller. This function provides a series of functions that includes logging and dump creation.

```
#define INCL_FFST
#include <os2.h>

PPRODUCTINFO    pProductInfo;
PFFSTPARMS      pFFSTParms;
APIRET          rc;          /* Return code. */

rc = FFSTProbe(pProductInfo, pFFSTParms);
```

FFSTProbe Parameter - pProductInfo

pProductInfo ([PPRODUCTINFO](#)) - input
Pointer to the product-information packet.

FFSTProbe Parameter - pFFSTParms

pFFSTParms (**PFFSTPARMS**) - input

A parameter packet that describes the information provided with the FFSTProbe.

FFSTProbe Return Value - rc

rc (APIRET) - returns

Return code.

FFSTProbe returns one of the following values:

- NO_ERROR
 - FFST_INVALID_PRODUCT_ADDRESS
 - FFST_INVALID_FFST_ADDRESS
 - FFST_INVALID_PRODDATA_ADDRESS
 - FFST_INVALID_DMI_ADDRESS
 - FFST_INVALID_DUMP_ADDRESS
 - FFST_INVALID_USER_AREA_ADDRESS
 - FFST_INVALID_LOG_DATA_ADDRESS
 - FFST_INVALID_MESSAGE_ADDRESS
 - FFST_INVALID_VENDOR_TAG_ADDRESS
 - FFST_INVALID_TAG_ADDRESS
 - FFST_INVALID_MODULENAME_ADDRESS
 - FFST_INVALID_REVISION_ADDRESS
 - FFST_INVALID_PRODUCT_ID_ADDRESS
 - FFST_INVALID_DMI_MODIFICATION_LEVEL_ADDRESS
 - FFST_INVALID_DMI_FIXLVL_ADDRESS
 - FFST_INVALID_TEMPFN_ADDRESS
 - FFST_INVALID_USER_STRUCTURE_TITLE_ADDRESS
 - FFST_INVALID_CONFIG_SUBPRODDATA_ADDRESS
 - FFST_INVALID_PRODUCT_REVISION
 - FFST_INVALID_DMI_REVISION
 - FFST_INVALID_FFST_REVISION
 - FFST_INVALID_SEVERITY
 - FFST_INVALID_USER_DUMP_NUMBER
 - FFST_INVALID_INSERTS_NUMBER
 - FFST_INVALID_INSERT_TEXT_ADDRESS
 - FFST_INVALID_PROBE_FLAGS
 - FFST_INVALID_PSTAT_DATA
 - FFST_INVALID_PROCESS_ENV
 - FFST_INVALID_PRODUCTDATA_PACKET_SIZE
 - FFST_INVALID_DMIDATA_PACKET_SIZE
 - FFST_INVALID_FFSTPARMS_PACKET_SIZE
 - FFST_DLL_QUERY_PROC_ERROR
 - FFST_DLL_LOAD_ERROR
 - FFST_MULTIPLE_SYSTEM_ERRORS
 - FFST_NOT_ACTIVE
 - FFST_GET_SHARED_MEM_ERROR
 - FFST_ALLOC_SHARED_MEM_ERROR
 - FFST_FREE_SHARED_MEM_ERROR
 - FFST_SEMAPHORE_TIMEOUT_ERROR
 - FFST_SEMAPHORE_OPEN_ERROR
 - FFST_SEMAPHORE_RELEASE_ERROR
 - FFST_SEMAPHORE_CLOSE_ERROR
 - FFST_SEMAPHORE_REQUEST_ERROR
-

FFSTProbe - Parameters

pProductInfo (**PPRODUCTINFO**) - input
Pointer to the product-information packet.

pFFSTParms (**FFSTPARMS**) - input
A parameter packet that describes the information provided with the FFSTProbe.

rc (**APIRET**) - returns
Return code.

FFSTProbe returns one of the following values:

- NO_ERROR
- FFST_INVALID_PRODUCT_ADDRESS
- FFST_INVALID_FFST_ADDRESS
- FFST_INVALID_PRODDATA_ADDRESS
- FFST_INVALID_DMI_ADDRESS
- FFST_INVALID_DUMP_ADDRESS
- FFST_INVALID_USER_AREA_ADDRESS
- FFST_INVALID_LOG_DATA_ADDRESS
- FFST_INVALID_MESSAGE_ADDRESS
- FFST_INVALID_VENDOR_TAG_ADDRESS
- FFST_INVALID_TAG_ADDRESS
- FFST_INVALID_MODULENAME_ADDRESS
- FFST_INVALID_REVISION_ADDRESS
- FFST_INVALID_PRODUCT_ID_ADDRESS
- FFST_INVALID_DMI_MODIFICATION_LEVEL_ADDRESS
- FFST_INVALID_DMI_FIXLVL_ADDRESS
- FFST_INVALID_TEMPFN_ADDRESS
- FFST_INVALID_USER_STRUCTURE_TITLE_ADDRESS
- FFST_INVALID_CONFIG_SUBPRODDATA_ADDRESS
- FFST_INVALID_PRODUCT_REVISION
- FFST_INVALID_DMI_REVISION
- FFST_INVALID_FFST_REVISION
- FFST_INVALID_SEVERITY
- FFST_INVALID_USER_DUMP_NUMBER
- FFST_INVALID_INSERTS_NUMBER
- FFST_INVALID_INSERT_TEXT_ADDRESS
- FFST_INVALID_PROBE_FLAGS
- FFST_INVALID_PSTAT_DATA
- FFST_INVALID_PROCESS_ENV
- FFST_INVALID_PRODUCTDATA_PACKET_SIZE
- FFST_INVALID_DMIDATA_PACKET_SIZE
- FFST_INVALID_FFSTPARMS_PACKET_SIZE
- FFST_DLL_QUERY_PROC_ERROR
- FFST_DLL_LOAD_ERROR
- FFST_MULTIPLE_SYSTEM_ERRORS
- FFST_NOT_ACTIVE
- FFST_GET_SHARED_MEM_ERROR
- FFST_ALLOC_SHARED_MEM_ERROR
- FFST_FREE_SHARED_MEM_ERROR
- FFST_SEMAPHORE_TIMEOUT_ERROR
- FFST_SEMAPHORE_OPEN_ERROR
- FFST_SEMAPHORE_RELEASE_ERROR
- FFST_SEMAPHORE_CLOSE_ERROR
- FFST_SEMAPHORE_REQUEST_ERROR

FFSTProbe - Remarks

The library LIBFFST.A must be linked with object files that use FFSTProbe.

FFSTProbe - Example Code

The following example adds an error (235) to the Default Log file for this service. The error has no message inserts, user data, dump file, or

extra data. Product information will be retrieved from the DMI database entry that was created when the product (TEST PRODUCT) was installed.

```
#define INCL_FFST
#include <unichar.h>
#include <os2.h>
PRODUCTINFO product_info_packet;
PRODUCTDATA product_data_packet;
MODINFO mod_info_packet;

.
.      (In main program)
.
PPRODUCTINFO pproduct_info_packet;
PPRODUCTDATA pproduct_data_packet;
PMODINFO pmod_info_packet;
APIRET      rc;

UniChar ibm[4] = L"IBM";
UniChar product[13] = L"TEST PRODUCT";
UniChar product_version_number[7] = L"V01R00";
UniChar subproduct[16] = L"TEST SUBPRODUCT";
UniChar module[16] = L"Module number 1";

pproduct_data_packet = &product_data_packet;
product_data_packet.packet_size = sizeof(PRODUCTDATA); /* size of packet */
product_data_packet.packet_revision_number =
        PRODUCTDATA_REVISION_NUMBER_1;
product_data_packet.product_manufacturer_name = &ibm; /* manufacture name */
product_data_packet.product_name = &product; /* product name */
product_data_packet.product_version_number =
        &product_version_number;
        /* product version number */

pproduct_info_packet = &product_info_packet;
product_info_packet.pProductData = pproduct_data_packet; /* Set pointer to product data */
product_info_packet.pDMI_Data = NULL; /* Retrieve data from DMI */
pmod_info_packet = &mod_info_packet;
mod_info_packet.packet_size = sizeof(MODINFO); /* size of packet */
mod_info_packet.packet_revision_number =
        MODINFO_REVISION_NUMBER_1;
mod_info_packet.subproduct = &subproduct; /* subproduct */
mod_info_packet.module_name = &module; /* module name */

.
.      (In Probe create routine)
.
FFSTPARMS FFSTParms_data_packet;
PFFSTPARMS pFFSTParms_data_packet;
pFFSTParms_data_packet = &FFSTParms_data_packet;

FFSTParms_data_packet.packet_size = sizeof(FFSTPARMS); /* packet size */

FFSTParms_data_packet.packet_revision_number =
        FFSTPARMS_WPOS_Revision_Number_1; /* revision number */
FFSTParms_data_packet.probe_ID = 1; /* probe ID number */
FFSTParms_data_packet.severity = severity3; /* severity */
FFSTParms_data_packet.template_record_ID = 235; /* error number */
product_data_packet.MSGINSDATA = NULL; /* Insert Data */
FFSTParms_data_packet.probe_Flags = 0; /* No probe Flags */
product_data_packet.DUMPUSEDATA = NULL; /* User Data */
FFSTParms_data_packet.log_user_data = NULL; /* No extra data */

rc = FFSTProbe(pproduct_info_packet, /* Product information */
               pmod_info_packet, /* Module information */
               pFFSTParms_data_packet) /* FFSTParms packet */

if (rc != 0)
{
    printf("FFSTProbe error: return code = %ld", rc);
    return;
}
```

FFSTProbe - Topics

Select an item:

FFSTQueryConfiguration

FFSTQueryConfiguration - Syntax

FFSTQueryConfiguration queries the FFST configuration parameters.

```
#define INCL_FFST
#include <os2.h>

PULONG          buffer_length;
PCONFIGPARMS    pConfigParms;
APIRET          rc;          /* Return code. */

rc = FFSTQueryConfiguration(buffer_length,
                             pConfigParms);
```

FFSTQueryConfiguration Parameter - buffer_length

buffer_length (PULONG) - in/out
Length (in bytes) of the buffer.

- On input, *buffer_length* contains the length of the provided buffer.
 - On output, *buffer_length* contains the total number of bytes that were placed in the buffer. If your buffer is too small, then *buffer_length* will be set to the required size, and no data will be written to the buffer.
-

FFSTQueryConfiguration Parameter - pConfigParms

pConfigParms (PCONFIGPARMS) - input
Pointer to the buffer where the FFST configuration parameters will be written.

FFSTQueryConfiguration Return Value - rc

rc (APIRET) - returns
Return code.

FFSTQueryConfiguration returns the following values:

- NO_ERROR
- FFST_INVALID_CONFIG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_NUMBER
- FFST_INVALID_CONFIG_PACKET_SIZE
- FFST_INVALID_CONFIG_DUMP_FILE_WRAP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_SIZE
- FFST_INVALID_CONFIG_KEEP_DUP_DUMP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_ADDRESS
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_LENGTH
- FFST_INVALID_CONFIG_NO_DISABLED_PRODUCTS
- FFST_INVALID_CONFIG_VENDOR_TAG_ADDRESS
- FFST_INVALID_CONFIG_TAG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_ADDRESS
- FFST_INVALID_CONFIG_MESSAGE_POPUP
- FFST_CONFIG_FILE_OPEN_ERROR
- FFST_CONFIG_FILE_SEARCH_ERROR
- FFST_CONFIG_FILE_WRITE_ERROR
- FFST_CONFIG_INSUFFICIENT_BUFFER
- FFST_INVALID_CONFIG_DUMP_FILE_DIR_DRIVE
- FFST_INVALID_DRIVE_REQUESTED
- FFST_INVALID_PATH_REQUESTED
- FFST_GET_SHARED_MEM_ERROR
- FFST_ALLOC_SHARED_MEM_ERROR
- FFST_FREE_SHARED_MEM_ERROR
- FFST_SEMAPHORE_TIMEOUT_ERROR
- FFST_SEMAPHORE_OPEN_ERROR
- FFST_SEMAPHORE_RELEASE_ERROR
- FFST_SEMAPHORE_CLOSE_ERROR
- FFST_SEMAPHORE_REQUEST_ERROR

FFSTQueryConfiguration - Parameters

buffer_length (PULONG) - in/out
Length (in bytes) of the buffer.

- On input, *buffer_length* contains the length of the provided buffer.
- On output, *buffer_length* contains the total number of bytes that were placed in the buffer. If your buffer is too small, then *buffer_length* will be set to the required size, and no data will be written to the buffer.

pConfigParms (PCONFIGPARMS) - input
Pointer to the buffer where the FFST configuration parameters will be written.

rc (APIRET) - returns
Return code.

FFSTQueryConfiguration returns the following values:

- NO_ERROR
- FFST_INVALID_CONFIG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_NUMBER
- FFST_INVALID_CONFIG_PACKET_SIZE
- FFST_INVALID_CONFIG_DUMP_FILE_WRAP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_SIZE
- FFST_INVALID_CONFIG_KEEP_DUP_DUMP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_ADDRESS
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_LENGTH
- FFST_INVALID_CONFIG_NO_DISABLED_PRODUCTS
- FFST_INVALID_CONFIG_VENDOR_TAG_ADDRESS
- FFST_INVALID_CONFIG_TAG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_ADDRESS
- FFST_INVALID_CONFIG_MESSAGE_POPUP
- FFST_CONFIG_FILE_OPEN_ERROR
- FFST_CONFIG_FILE_SEARCH_ERROR

- FFST_CONFIG_FILE_WRITE_ERROR
- FFST_CONFIG_INSUFFICIENT_BUFFER
- FFST_INVALID_CONFIG_DUMP_FILE_DIR_DRIVE
- FFST_INVALID_DRIVE_REQUESTED
- FFST_INVALID_PATH_REQUESTED
- FFST_GET_SHARED_MEM_ERROR
- FFST_ALLOC_SHARED_MEM_ERROR
- FFST_FREE_SHARED_MEM_ERROR
- FFST_SEMAPHORE_TIMEOUT_ERROR
- FFST_SEMAPHORE_OPEN_ERROR
- FFST_SEMAPHORE_RELEASE_ERROR
- FFST_SEMAPHORE_CLOSE_ERROR
- FFST_SEMAPHORE_REQUEST_ERROR

FFSTQueryConfiguration - Remarks

The library LIBFFST.A must be linked with object files that use FFSTQueryConfiguration

FFSTQueryConfiguration - Related Functions

- [FFSTSetConfiguration](#)
-

FFSTQueryConfiguration - Example Code

The following example will query for FFST configuration. FFST configuration values are returned in the buffer. You can access individual parameters by using the [CONFIGPARMS](#) structure.

```
#define INCL_FFST
#include <unichar.h>
#include <os2.h>
APIRET rc;
ULONG buffer_length;
UniChar config_buffer[300];

CONFIGPARMS  FFSTConfig;
PCONFIGPARMS pFFSTConfig  *FFSTConfig;

buffer_length = 300;
rc = FFSTQueryConfiguration (  &buffer_length, &config_buffer );
if (rc != 0)
{
    /* If Problem          */
    /* reason              */
    printf("FFSTConfigure error: return code = %d",rc);
    return;
}
```

FFSTQueryConfiguration - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

FFSTSetConfiguration

FFSTSetConfiguration - Syntax

FFSTSetConfiguration sets the FFST configuration parameters.

You should use [FFSTQueryConfiguration](#) to obtain the current settings before using this function. This allows you to change the desired parameters without affecting others.

```
#define INCL_FFST
#include <os2.h>

PCONFIGPARMS    pConfigParms;
APIRET          rc;          /* Return code. */

rc = FFSTSetConfiguration(pConfigParms);
```

FFSTSetConfiguration Parameter - pConfigParms

pConfigParms ([PCONFIGPARMS](#)) - input
Pointer to FFST configuration parameters.

FFSTSetConfiguration Return Value - rc

rc (APIRET) - returns
Return code.

FFSTSetConfiguration returns the following values:

- NO_ERROR
- FFST_INVALID_CONFIG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_NUMBER
- FFST_INVALID_CONFIG_PACKET_SIZE
- FFST_INVALID_CONFIG_DUMP_FILE_WRAP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_SIZE
- FFST_INVALID_CONFIG_KEEP_DUP_DUMP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_ADDRESS
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_LENGTH
- FFST_INVALID_CONFIG_NO_DISABLED_PRODUCTS
- FFST_INVALID_CONFIG_VENDOR_TAG_ADDRESS
- FFST_INVALID_CONFIG_TAG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_ADDRESS
- FFST_INVALID_CONFIG_MESSAGE_POPUP

- FFST_CONFIG_FILE_OPEN_ERROR
- FFST_CONFIG_FILE_SEARCH_ERROR
- FFST_CONFIG_FILE_WRITE_ERROR
- FFST_CONFIG_INSUFFICIENT_BUFFER
- FFST_INVALID_CONFIG_DUMP_FILE_DIR_DRIVE
- FFST_INVALID_DRIVE_REQUESTED
- FFST_INVALID_PATH_REQUESTED
- FFST_GET_SHARED_MEM_ERROR
- FFST_ALLOC_SHARED_MEM_ERROR
- FFST_FREE_SHARED_MEM_ERROR
- FFST_SEMAPHORE_TIMEOUT_ERROR
- FFST_SEMAPHORE_OPEN_ERROR
- FFST_SEMAPHORE_RELEASE_ERROR
- FFST_SEMAPHORE_CLOSE_ERROR
- FFST_SEMAPHORE_REQUEST_ERROR

FFSTSetConfiguration - Parameters

pConfigParms ([PCONFIGPARMS](#)) - input

Pointer to FFST configuration parameters.

rc ([APIRET](#)) - returns

Return code.

FFSTSetConfiguration returns the following values:

- NO_ERROR
- FFST_INVALID_CONFIG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_NUMBER
- FFST_INVALID_CONFIG_PACKET_SIZE
- FFST_INVALID_CONFIG_DUMP_FILE_WRAP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_SIZE
- FFST_INVALID_CONFIG_KEEP_DUP_DUMP
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_ADDRESS
- FFST_INVALID_CONFIG_DUMP_FILE_DIRECTORY_NAME_LENGTH
- FFST_INVALID_CONFIG_NO_DISABLED_PRODUCTS
- FFST_INVALID_CONFIG_VENDOR_TAG_ADDRESS
- FFST_INVALID_CONFIG_TAG_ADDRESS
- FFST_INVALID_CONFIG_REVISION_ADDRESS
- FFST_INVALID_CONFIG_MESSAGE_POPUP
- FFST_CONFIG_FILE_OPEN_ERROR
- FFST_CONFIG_FILE_SEARCH_ERROR
- FFST_CONFIG_FILE_WRITE_ERROR
- FFST_CONFIG_INSUFFICIENT_BUFFER
- FFST_INVALID_CONFIG_DUMP_FILE_DIR_DRIVE
- FFST_INVALID_DRIVE_REQUESTED
- FFST_INVALID_PATH_REQUESTED
- FFST_GET_SHARED_MEM_ERROR
- FFST_ALLOC_SHARED_MEM_ERROR
- FFST_FREE_SHARED_MEM_ERROR
- FFST_SEMAPHORE_TIMEOUT_ERROR
- FFST_SEMAPHORE_OPEN_ERROR
- FFST_SEMAPHORE_RELEASE_ERROR
- FFST_SEMAPHORE_CLOSE_ERROR
- FFST_SEMAPHORE_REQUEST_ERROR

FFSTSetConfiguration - Remarks

The library LIBFFST.A must be linked with object files that use FFSTSetConfiguration

FFSTSetConfiguration - Related Functions

- [FFSTQueryConfiguration](#)

FFSTSetConfiguration - Example Code

The following example will instruct FFST to stop wrapping dumps. The rest of the parameters are left to their earlier values.

```
#define INCL_FFST
#include <unichar.h>
#include <os2.h>
APIRET rc;

CONFIGPARMS  FFSTConfig;
PCONFIGPARMS pFFSTConfig  *FFSTConfig;

FFSTConfig.dump_file_wrap = FFST_DUMP_WRAP_OFF; /* Set dump wrap to off*/
/* set rest of the parameters to indicate no change */

FFSTConfig.dump_file_directory_size = FFST_DUMP_FILE_DIRECTORY_SIZE_NO_CHANGE;
FFSTConfig.keep_dup_dump = FFST_KEEP_DUP_DUMP_NO_CHANGE;
FFSTConfig.dump_file_directory_length = FFST_DUMP_FILE_DIRECTORY_LENGTH_NO_CHANGE;
FFSTConfig.no_of_disabled_products = FFST_NO_OF_PROBE_DISABLED_PRODUCTS_NO_CHANGE;
FFSTConfig.dump_file_directory = NULL;
FFSTConfig.PProductData = NULL;

rc = FFSTSetConfiguration( pFFSTConfig);
if (rc != 0)
{
    /* If Problem */
    /* reason */
    printf("FFSTConfigure error: return code = %d",rc);
    return;
}
```

FFSTSetConfiguration - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

LogChangeEventFilter

LogChangeEventFilter - Syntax

LogChangeEventFilter is used to alter the event-notification filter that is associated with an event-notification request. In addition to changing the filter, you can specify current event-notification entries that are to be purged before the filter change takes effect.

```
#define INCL_LOGGING
#include <os2.h>

ULONG      service;
PVOID      pChangeEventFilter;
APIRET     rc;

rc = LogChangeEventFilter(service, pChangeEventFilter);
```

LogChangeEventFilter Parameter - service

service (ULONG) - input
The class of Logging Service:

- | | |
|---|---|
| 1 | Error logging |
| | All other values are reserved for future use. |

LogChangeEventFilter Parameter - pChangeEventFilter

pChangeEventFilter (PVOID) - in/out
A pointer to the LogChangeEventFilter parameter packet.
For Error Logging, this is a pointer to a [LCEFREQUEST](#) structure.

LogChangeEventFilter Return Value - rc

rc (APIRET) - returns
Return code.

LogChangeEventFilter returns the following values:

- NO_ERROR
- ERROR_LF_INVALID_SERVICE
- INVALID_DATA_POINTER
- INVALID_LF_PACKET_REVISION_NUMBER
- RAS_INVALID_PARM_PACKET_PTR
- INVALID_LF_FLAG
- RAS_INVALID_LOG_NOTIFY_ID
- RAS_INVALID_PACKET_SIZE
- RAS_NOTIF_ENTRY_NOT_FOUND
- RAS_NOTIF_ENTRY_DELETED
- RAS_ENTRY_FILTER_UNCHANGED

LogChangeEventFilter - Parameters

service (ULONG) - input

The class of Logging Service:

1 Error logging

All other values are reserved for future use.

pChangeEventFilter (PVOID) - in/out

A pointer to the LogChangeEventFilter parameter packet.

For Error Logging, this is a pointer to a [LCEFREQUEST](#) structure.

rc (APIRET) - returns

Return code.

LogChangeEventFilter returns the following values:

- NO_ERROR
- ERROR_LF_INVALID_SERVICE
- INVALID_DATA_POINTER
- INVALID_LF_PACKET_REVISION_NUMBER
- RAS_INVALID_PARM_PACKET_PTR
- INVALID_LF_FLAG
- RAS_INVALID_LOG_NOTIFY_ID
- RAS_INVALID_PACKET_SIZE
- RAS_NOTIF_ENTRY_NOT_FOUND
- RAS_NOTIF_ENTRY_DELETED
- RAS_ENTRY_FILTER_UNCHANGED

LogChangeEventFilter - Remarks

The library LIBLFAPL.A must be linked with object files that use LogChangeEventFilter

LogChangeEventFilter - Related Functions

- [LogOpenEventNotification](#)
 - [LogCloseEventNotification](#)
 - [LogWaitEvent](#)
 - [LogReadEntry](#)
-

LogChangeEventFilter - Example Code

The following example changes the event-notification filter for an event notification to a NULL filter (that is, any Error Log entry that is logged will cause an event notification to be sent). The sample will also purge any event notifications that might be pending at the time the LogChangeEventFilter call is made.

```
#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>
.
{
    APIRET rc;                               /* return code */
    ULONG service;
    LCEFREQUEST change_event_filter_packet;
```

```

HLOGNOTIFY log_notify;

service = ERROR_LOGGING_SERVICE;

/* Construct the LogChangeEventFilter parameter packet */
change_event_filter_packet.packet_size = sizeof(LCEFREQUEST);
change_event_filter_packet.packet_revision_number = WPOS_RELEASE_1;
change_event_filter_packet.purge_flags = PURGE_EVENT_NOTIFICATION;
change_event_filter_packet.LogNotify = log_notify;
change_event_filter_packet.pFilter = NULL;

rc = LogChangeEventFilter(service, /* service */
                        &change_event_filter_packet) /* parameter packet*/
if (rc != 0)
{
    printf("LogChangeEventFilter error: return code = %d",rc);
    return;
}

```

LogChangeEventFilter - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

LogCloseEventNotification

LogCloseEventNotification - Syntax

LogCloseEventNotification closes event-notification requests.

```

#define INCL_LOGGING
#include <os2.h>

ULONG      service;
PVOID      pCloseEventNotification;
APIRET     rc;

rc = LogCloseEventNotification(service, pCloseEventNotification);

```

LogCloseEventNotification Parameter - service

service (ULONG) - input

The class of Logging Service:

1 Error logging

All other values are reserved for future use.

LogCloseEventNotification Parameter - pCloseEventNotification

pCloseEventNotification (PVOID) - in/out

A pointer to the LogCloseEventNotification parameter packet.

For Error Logging, this is a pointer to a [LCENREQUEST](#) structure.

LogCloseEventNotification Return Value - rc

rc (APIRET) - returns

Return code.

LogCloseEventNotification returns the following values:

- NO_ERROR
 - ERROR_LF_INVALID_SERVICE
 - INVALID_DATA_POINTER
 - INVALID_LF_PACKET_REVISION_NUMBER
 - RAS_INVALID_PARM_PACKET_PTR
 - RAS_INVALID_LOG_NOTIFY_ID
 - RAS_INVALID_PACKET_SIZE
 - RAS_INTERNAL_MEMORY_FAILURE
 - RAS_NOTIF_ENTRY_NOT_FOUND
-

LogCloseEventNotification - Parameters

service (ULONG) - input

The class of Logging Service:

1 Error logging

All other values are reserved for future use.

pCloseEventNotification (PVOID) - in/out

A pointer to the LogCloseEventNotification parameter packet.

For Error Logging, this is a pointer to a [LCENREQUEST](#) structure.

rc (APIRET) - returns

Return code.

LogCloseEventNotification returns the following values:

- NO_ERROR
- ERROR_LF_INVALID_SERVICE
- INVALID_DATA_POINTER
- INVALID_LF_PACKET_REVISION_NUMBER
- RAS_INVALID_PARM_PACKET_PTR
- RAS_INVALID_LOG_NOTIFY_ID

- [RAS_INVALID_PACKET_SIZE](#)
- [RAS_INTERNAL_MEMORY_FAILURE](#)
- [RAS_NOTIF_ENTRY_NOT_FOUND](#)

LogCloseEventNotification - Remarks

The library LIBLFAPI.A must be linked with object files that use LogCloseEventNotification

LogCloseEventNotification - Related Functions

- [LogOpenEventNotification](#)
- [LogChangeEventFilter](#)
- [LogWaitEvent](#)
- [LogReadEntry](#)

LogCloseEventNotification - Example Code

The following example closes an event-notification mechanism that is connected to the Error Logging service.

```
#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>

.
.
.

{
    APIRET rc;                /* return code */
    ULONG service;
    LCENREQUEST close_event_packet;
    HLOGNOTIFY log_notify;

    service = ERROR_LOGGING_SERVICE;

    /* Construct the LogChangeEventFilter parameter packet */
    close_event_packet.packet_size = sizeof(LCENREQUEST);
    close_event_packet.packet_revision_number = WPOS_RELEASE_1;
    close_event_packet.LogNotify = log_notify;
    rc = LogCloseEventNotification(service,      /* service */
                                   &close_event_packet) /* parameter packet */
    if (rc != 0)
    {
        printf("LogCloseEventNotification error: return code = %d",rc);
        return;
    }
}
```

LogCloseEventNotification - Topics

Select an item:

LogCloseFile

LogCloseFile - Syntax

LogCloseFile closes a file that was previously opened by [LogOpenFile](#).

```
#define INCL_LOGGING
#include <os2.h>

ULONG      service;
PVOID      pCloseFile;
APIRET     rc;

rc = LogCloseFile(service, pCloseFile);
```

LogCloseFile Parameter - service

service (ULONG) - input
The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error logging |
| 2 | Event tracing |

All other values are reserved for future use.

LogCloseFile Parameter - pCloseFile

pCloseFile (PVOID) - input
A pointer to the LogCloseFile parameter packet.

For Error Logging, this is a pointer to a [LCFREQUEST](#) structure.

For Event Tracing, this is a pointer to a [LCFREQUESTTRACE](#) structure.

LogCloseFile Return Value - rc

rc (APIRET) - returns
Return code.

LogCloseFile returns one of the following values:

- NO_ERROR
- ERROR_LF_BUF_TOO_SMALL
- ERROR_ACCESS_DENIED
- ERROR_LF_INVALID_SERVICE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_DATA_POINTER
- INVALID_LF_PARM_PACKET_PTR
- ERROR_LF_INVALID_PACKET_SIZE

LogCloseFile - Parameters

service (ULONG) - input
The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error logging |
| 2 | Event tracing |

All other values are reserved for future use.

pCloseFile (PVOID) - input
A pointer to the LogCloseFile parameter packet.

For Error Logging, this is a pointer to a [LCFREQUEST](#) structure.

For Event Tracing, this is a pointer to a [LCFREQUESTTRACE](#) structure.

rc (APIRET) - returns
Return code.

LogCloseFile returns one of the following values:

- NO_ERROR
- ERROR_LF_BUF_TOO_SMALL
- ERROR_ACCESS_DENIED
- ERROR_LF_INVALID_SERVICE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_DATA_POINTER
- INVALID_LF_PARM_PACKET_PTR
- ERROR_LF_INVALID_PACKET_SIZE

LogCloseFile - Remarks

The library LIBLFAPI.A must be linked with object files that use LogCloseFile

LogCloseFile - Related Functions

- [LogOpenFile](#)
- [LogReadEntry](#)

LogCloseFile - Example Code

The following example closes a log file (log_file_ID = 2) that was opened with [LogOpenFile](#).

```
#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>
.
.
.
{
    APIRET rc;                                /* return code */
    ULONG service;
    LCFREQUEST close_file_packet;

    service = ERROR_LOGGING_SERVICE;

    /* Construct the LogOpenFile parameter packet */
    close_file_packet.packet_size = sizeof(LCFREQUEST);
    close_file_packet.packet_revision_number = WPOS_RELEASE_1;
    close_file_packet.log_file_ID = 2;

    rc = LogCloseFile(service,                  /* service */
                      &close_file_packet)      /* parameter packet */
    if (rc != 0)
    {
        printf("LogCloseFile error: return code = %d",rc);
        return;
    }
}
```

LogCloseFile - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

LogFormatEntry

LogFormatEntry - Syntax

LogFormatEntry formats a Log Entry for display.

```
#define INCL_LOGGING
#include <os2.h>

ULONG      service;
PVOID      pFormatEntry;
APIRET     rc;

rc = LogFormatEntry(service, pFormatEntry);
```

LogFormatEntry Parameter - service

service (ULONG) - input

The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error logging |
| 2 | Event tracing |

All other values are reserved for future use.

LogFormatEntry Parameter - pFormatEntry

pFormatEntry (PVOID) - input

A pointer to the LogFormatEntry parameter packet.

For Error Logging, this is a pointer to a [LFFERREQUEST](#) structure.

For Event Tracing, this is a pointer to a [LFFERREQUESTTRACE](#) structure.

LogFormatEntry Return Value - rc

rc (APIRET) - returns

Return code.

LogFormatEntry returns the following values:

- NO_ERROR
- ERROR_FILE_NOT_FOUND
- ERROR_LF_BUF_TOO_SMALL
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_DATA_POINTER
- INVALID_LF_PARM_PACKET_PTR
- INVALID_LF_FLAG
- ERROR_LF_INVALID_PACKET_SIZE
- INVALID_LOG_ENTRY_RECORD
- NO_LOG_ENTRY_FORMAT_TEMPLATE_AVAILABLE
- RAS_UNICODE_CONVERSION_ERROR
- RAS_INVALID_LOCALE_OBJECT

LogFormatEntry - Parameters

service (ULONG) - input

The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error logging |
| 2 | Event Tracing |

All other values are reserved for future use.

pFormatEntry (PVOID) - input

A pointer to the LogFormatEntry parameter packet.

For Error Logging, this is a pointer to a [LFFERREQUEST](#) structure.

For Event Tracing, this is a pointer to a [LFFERREQUESTTRACE](#) structure.

rc (APIRET) - returns

Return code.

LogFormatEntry returns the following values:

- NO_ERROR
- ERROR_FILE_NOT_FOUND
- ERROR_LF_BUF_TOO_SMALL
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_DATA_POINTER
- INVALID_LF_PARM_PACKET_PTR
- INVALID_LF_FLAG
- ERROR_LF_INVALID_PACKET_SIZE
- INVALID_LOG_ENTRY_RECORD
- NO_LOG_ENTRY_FORMAT_TEMPLATE_AVAILABLE
- RAS_UNICODE_CONVERSION_ERROR
- RAS_INVALID_LOCALE_OBJECT

LogFormatEntry - Remarks

ADDITIONAL RETURNS INFORMATION

The data will be passed back in multiple occurrences of the following LTD (Length, Type, Data) format:

ULONG	length;
ULONG	type;
UniChar	data[n];

Where:

length (ULONG) is the length, in bytes, of this detail record (includes length, type, and data fields).

type (ULONG) is an integer value that represents the type of data being passed back. The following are the current Error Logging type values and their meanings:

Type	Meaning
0005	Date heading
0006	Date
0007	Time heading

0008	Time
0009	Entry ID heading
0010	Entry ID
0011	Severity heading
0012	Severity
0013	Module name heading
0014	Module name
0015	Directory name heading
0016	Directory name
0017	Error message heading
0018	Error message text
0019	Probe ID heading
0020	Probe ID text
0021	Probe Flags heading
0022	Probe Flags
0023	Template Repository pathname heading
0024	Template Repository pathname text
0025	Template ID heading
0026	Template ID text
0027	Dump generated heading
0028	Dump Generated text
0029	Trace file generated heading
0030	Trace File generated text
0031	Process dump generated heading
0032	Process Dump generated text
0040	Failure Causes heading
0041	Failure Cause (Could be 4 of these)
0050	Failure Actions heading
0051	Failure Action (Could be 4 of these)
0060	Install Causes heading
0061	Install Cause (Could be 4 of these)
0070	Install Actions heading
0071	Install Action (Could be 4 of these)
0080	User Causes heading
0081	User Cause (Could be 4 of these)
0090	User Actions heading
0091	User Action (Could be 4 of these)
0100	Return Code heading

0101	Return Code text
0110	Dump File name heading
0111	Dump File name text
0112	Dump formatter heading
0113	Dump Formatter text
0114	Dump File Deletion Date heading
0115	Dump File Deletion Date
0116	Dump File Deletion Time heading
0117	Dump File Deletion Time
0120	Trace File name heading
0121	Trace File name text
0122	Trace formatter heading
0123	Trace formatter text
0124	Trace File Deletion Date heading
0125	Trace File Deletion Date
0126	Trace File Deletion Time heading
0127	Trace File Deletion Time
0130	Process Dump File name heading
0131	Process Dump File name text
0132	Process Dump formatter heading
0133	Process Dump formatter text
0134	Process DumpDeletion Date heading
0135	Process Dump File Deletion Date
0136	Process Dump Deletion Time heading
0137	Process Dump File Deletion Time
0140	PCT heading
0141	PCT Execution Parameters
0150	DMI vendor tag heading
0151	DMI vendor tag text
0155	DMI tag heading
0156	DMI tag text
0165	DMI product ID heading
0166	DMI product ID text
0170	DMI revision heading
0171	DMI revision text
0172	DMI modification level heading
0173	DMI modification level text
0174	DMI fix level heading

0175	DMI fix level text
0195	Machine type heading
0196	Machine type text
0200	Machine serial number heading
0201	Machine serial number text
0205	Hostname heading
0206	Hostname text
0210	User Data heading
0211	User data text
0213	Action heading (DET2 record)
0214	Action text (DET2 record)
0215	Old Value heading (DET2 record)
0216	Old Value text (DET2 record)
0220	New Value heading (DET2 record)
0221	New Value text (DET2 record)

The following are the Event Trace data types and their meanings:

Type	Meaning
7001	Time
7002	Entry ID
7003	Personality
7010	MicroKernel Task ID
7011	MicroKernel Thread Id
7012	OS/2 Process ID
7013	OS/2 Thread ID
7020	Module pathname
7030	Major code
7031	Minor code
7032	Entry data type field.
7040	Data 1
7041	Data 2
7042	Data 3
7043	Data 4
7044	Data 5
7045	Data 6
7050	date_logging_began
7051	time_logging_began
7052	date_logging_ended

7053	time_logging_ended
7060	log_file_size
7061	max_log_file_size
7070	comment_string
7080	system_trace_buffer_size
7081	system_hostname
7082	system_serial_number
7083	system_machine_type
7084	operating system version
7100	Event Description
7101	Buffer 1 (As many buffers as specified by users)
7110	Buffer 10

data (UniChar[]) A variable length area that contains the formatted data.

Note: A record will always be placed in the buffer for each of the requested types. If there is no data, only the length and type portion of the record will be returned. The length would indicate that there is no data.

The library LIBLFAPI.A must be linked with object files that use LogFormatEntry

LogFormatEntry - Related Functions

- [LogAppendEntries](#)
- [LogReadEntry](#)

LogFormatEntry - Example Code

The following example formats an Error Log record for display. The calling program has placed the address of the locale object in *locale*.

```
#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>
.
{
    APIRET rc;                                /* return code */
    ULONG service;
    LFEREQUEST format_entry_packet;
    BYTE log_entry_buffer[1024];
    UniChar string_buffer[2048];
    #define STRING_BUFFER_LENGTH 2048
    ULONG string_buffer_length;
    PVOID locale;

    service = ERROR_LOGGING_SERVICE;
    string_buffer_length = STRING_BUFFER_LENGTH;

    /* Construct the Error Log Service format packet */
    format_entry_packet.packet_size = sizeof(LFEREQUEST);
    format_entry_packet.packet_revision_number = WPOS_RELEASE_1;
    format_entry_packet.log_entry_buffer = &log_entry_buffer;
    format_entry_packet.flags = ERR_FORMAT_DETAIL_DATA;
    format_entry_packet.locale_object = locale;
    format_entry_packet.string_buffer_length = &string_buffer_length;
    format_entry_packet.string_buffer = &string_buffer;
```

```
rc = LogFormatEntry(service,          /* service */
                   &format_entry_packet) /* parameter packet */
if (rc != 0)
{
    printf("LogFormatEntry error: return code = %d",rc);
    return;
}
```

LogFormatEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

LogOpenEventNotification

LogOpenEventNotification - Syntax

LogOpenEventNotification registers a consumer with the Logging Service, so that the consumer will receive notification when specific log records have been created. Consumers specify which log records they will be notified about by providing filtering information. If no filter data structure is provided, all events that are logged to the specified log file will cause event notifications to be forwarded to the consumer. LogOpenEventNotification returns an ID used to reference this notification request.

[LogChangeEventFilter](#) can be used to change the filter that is in effect for a given registration.

[LogWaitEvent](#) is used to ask the Logging Service to send the next log record that meets the criteria specified in the registration. When an event notification is received, the consumer receives an event-key data structure and corresponding log entry.

[LogCloseEventNotification](#) is used to remove the registration so that no further notifications will be received by the consumer.

```
#define INCL_LOGGING
#include <os2.h>

ULONG          service;
PLOENREQUEST   pOpenEventNotification;
APIRET        rc;

rc = LogOpenEventNotification(service, pOpenEventNotification);
```

LogOpenEventNotification Parameter - service

service (ULONG) - input

The class of Logging Service:

1 Error logging

LogOpenEventNotification Parameter - pOpenEventNotification

pOpenEventNotification ([PLOENREQUEST](#)) - in/out

A pointer to the LogOpenEventNotification parameter packet.

For Error Logging, this is a pointer to a [LOENREQUEST](#) structure.

LogOpenEventNotification Return Value - rc

rc (APIRET) - returns

Return code.

LogOpenEventNotification returns the following values:

- NO_ERROR
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- RAS_INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- RAS_INVALID_PARM_PACKET_PTR
- RAS_INVALID_FLAG
- RAS_INVALID_NOTIFY_PRT
- RAS_INVALID_PACKET_SIZE

LogOpenEventNotification - Parameters

service (ULONG) - input

The class of Logging Service:

1 Error logging

pOpenEventNotification ([PLOENREQUEST](#)) - in/out

A pointer to the LogOpenEventNotification parameter packet.

For Error Logging, this is a pointer to a [LOENREQUEST](#) structure.

rc (APIRET) - returns

Return code.

LogOpenEventNotification returns the following values:

- NO_ERROR
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- RAS_INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- RAS_INVALID_PARM_PACKET_PTR
- RAS_INVALID_FLAG
- RAS_INVALID_NOTIFY_PRT

- RAS_INVALID_PACKET_SIZE

LogOpenEventNotification - Remarks

The event-notification filter is a flexible data structure that is used to specify the class of events whose notifications will be received through the event-notification mechanism. It is available to event consumers (through [LogOpenEventNotification](#) and [LogChangeEventFilter](#)) and to log file readers (through [LogReadEntry](#)). This provides a common search criteria when waiting for events and reading selected entries within a log file.

EVENT NOTIFICATION FILTER STRUCTURE

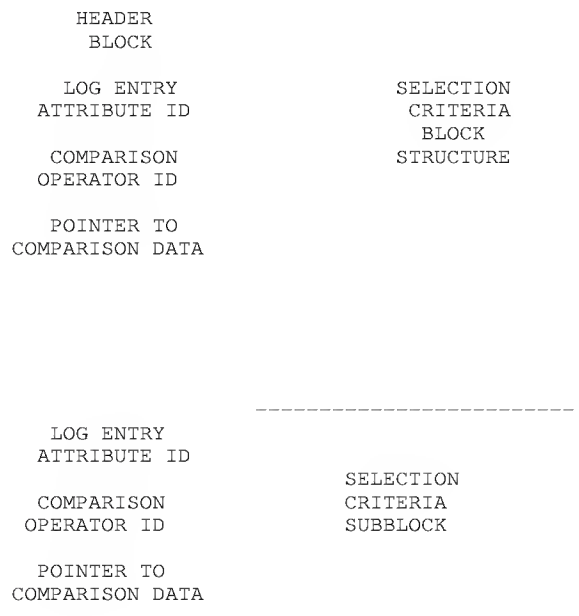
```
SELECTION CRITERIA    SELECTION CRITERIA    SELECTION CRITERIA
BLOCK                > BLOCK                > BLOCK
```

The event-notification filter consists of an array of one or more selection criteria blocks. Each selection criteria block contains a small header block that specifies the revision of the filter and points to the next selection criteria block.

Each selection criteria block consists of an array of selection criteria subblocks. Each selection criteria subblock contains three pieces of information:

1. The ID of an attribute that is contained within this class of log entry.
2. A comparison operator that is to be applied against the specified log entry attribute.
3. A pointer to a data value that is to be compared against the specified attribute within the log entry.

The following diagram summarizes the structure of a selection criteria block:

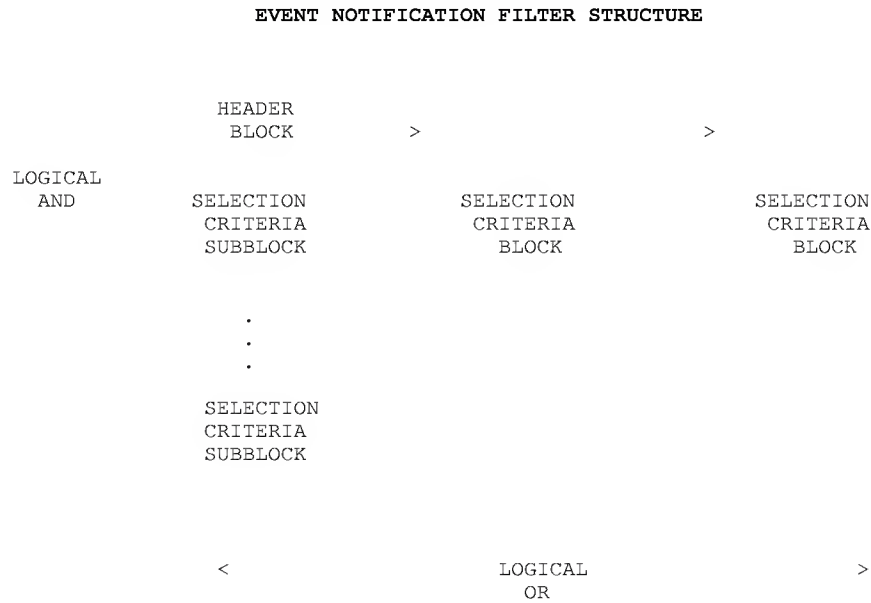


Each selection criteria subblock specifies a comparison with respect to an attribute within a log entry. The result of the comparison is a Boolean value. The Boolean value of the array of selection criteria subblocks (within a selection criteria block) is the logical AND of the Boolean values for each subblock.

If an event-notification filter contains more than one selection criteria block, the entire chain of selection criteria blocks is considered to resolve

to the logical OR of the Boolean values of the individual blocks. In this manner, a consumer can construct appropriately complex event-discrimination filters.

The following diagram illustrates the logical representation of an event-notification filter:



The library LIBLFAPI.A must be linked with object files that use LogOpenEventNotification

LogOpenEventNotification - Related Functions

- [LogQueryStateFile](#)
- [LogChangeEventFilter](#)
- [LogWaitEvent](#)
- [LogReadEntry](#)

LogOpenEventNotification - Example Code

The following example opens error-logging event notification. It will initialize an event-notification filter that specifies any Error Log entry that has a product manufacturer named "IBM" and a severity less than 4.

```

#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>
#define ERROR_LOG_FILE_ID 1

.
.
.

ULONG service;
ULONG severity = 4;
ULONG entry_id = 12;
LOENREQUEST open_event_packet;
HLOGNOTIFY log_notify;
SUBBLOCK subblock1, subblock2, subblock3;
HEADERBLOCK headerblock1, headerblock2;
FILTERBLOCK filter;

struct UniCode_manufacturer_name {

```

```

        UniChar manufacturer_name[4] = L"IBM";
    } manufacturer_name;

    service = ERROR_LOGGING_SERVICE;

    /* Construct an event notification filter with 2 header blocks. */
    /* The first header block points to a single subblock. */
    /* The second header block points to a chain of two subblocks. */

    filter.packet_size = sizeof(FILTERBLOCK);
    filter.packet_revision_number = WPOS_RELEASE_1;
    filter.header_block = &headerblock1;
    filter.header_blocks = 2;

    /*-----construct headerblock1-----*/
    headerblock1.pSubblock = &subblock1;
    headerblock1.pNextBlock = &headerblock2;

    /*construct subblock1 of headerblock1*/
    subblock1.entry_attribute_ID = LOG_ERROR_DMI_VENDOR_TAG;
    subblock1.comparison_operator = LOG_ERROR_EQUAL;
    subblock1.comparison_data_ptr = &manufacturer_name;
    subblock1.next_subblock = NULL;

    /*-----construct headerblock2-----*/
    headerblock2.pSubblock = &subblock2;
    headerblock2.pNextBlock = NULL;

    /*construct subblock2 of headerblock2*/
    subblock2.entry_attribute_ID = LOG_ERROR_SEVERITY;
    subblock2.comparison_operator = LOG_ERROR_LESS_THAN;
    subblock2.comparison_data_ptr = severity;
    subblock2.next_subblock = &subblock3;

    /*construct subblock3 of headerblock2*/
    subblock3.entry_attribute_ID = LOG_ERROR_ENTRY_ID;
    subblock3.comparison_operator = LOG_ERROR_GREATER_THAN;
    subblock3.comparison_data_ptr = entry_id;
    subblock3.next_subblock = null;

    /* Construct the LogOpenEventNotification parameter packet.*/
    open_event_packet.packet_size = sizeof(LOENREQUEST);
    open_event_packet.packet_revision_number = WPOS_RELEASE_1;
    open_event_packet.log_file_ID = ERROR_LOG_FILE_ID;
    open_event_packet.pLogNotify = &log_notify;
    open_event_packet.pFilter = &filter;
    open_event_packet.read_flags = 0;

    rc = LogOpenEventNotification(service, /*service*/
                                &open_event_packet); /*parameter packet*/

    if (rc != 0)
    {
        printf("LogOpenEventNotification error: return code = %d",rc);
        return;
    }

```

LogOpenEventNotification - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

LogOpenFile

LogOpenFile - Syntax

LogOpenFile opens a log file so that it can be read using [LogReadEntry](#). LogOpenFile returns a log_file_ID number that is used to refer to the file.

For Error Logging:

Specify either a log_file_ID number or a path name. If the log_file_ID number is specified and the log_file_ID is valid the path name is returned. If a path name is specified, the Logging Service will return the log_file_ID that corresponds to the file. (The file will be opened if it is not already open.)

For Event Tracing:

Specify a pathname for the file to be opened.

```
#define INCL_LOGGING
#include <os2.h>

ULONG      service;
PVOID      pOpenFile;
APIRET     rc;

rc = LogOpenFile(service, pOpenFile);
```

LogOpenFile Parameter - service

service (ULONG) - input

The class of Logging Service:

1	Error logging
2	Event tracing

All other values are reserved for future use.

LogOpenFile Parameter - pOpenFile

pOpenFile (PVOID) - in/out

A pointer to the LogOpenFile parameter packet.

For Error Logging, this is a pointer to a [LOFREQUEST](#) structure.

For Event Tracing, this is a pointer to a [LOFREQUESTTRACE](#) structure.

LogOpenFile Return Value - rc

rc (APIRET) - returns
Return code.

LogOpenFile returns the following values:

- NO_ERROR
- ERROR_FILE_NOT_FOUND
- ERROR_DEVICE_IN_USE
- ERROR_DRIVE_LOCKED
- ERROR_OPEN_FAILED
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_LF_FILENAME_LENGTH
- INVALID_LF_FILENAME_PTR
- INVALID_LF_PARM_PACKET_PTR
- INVALID_LF_LOG_FILE
- ERROR_LF_INVALID_PACKET_SIZE

LogOpenFile - Parameters

service (ULONG) - input
The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error logging |
| 2 | Event tracing |

All other values are reserved for future use.

pOpenFile (PVOID) - in/out
A pointer to the LogOpenFile parameter packet.

For Error Logging, this is a pointer to a [LOFREQUEST](#) structure.

For Event Tracing, this is a pointer to a [LOFREQUESTTRACE](#) structure.

rc (APIRET) - returns
Return code.

LogOpenFile returns the following values:

- NO_ERROR
- ERROR_FILE_NOT_FOUND
- ERROR_DEVICE_IN_USE
- ERROR_DRIVE_LOCKED
- ERROR_OPEN_FAILED
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_LF_FILENAME_LENGTH
- INVALID_LF_FILENAME_PTR
- INVALID_LF_PARM_PACKET_PTR
- INVALID_LF_LOG_FILE
- ERROR_LF_INVALID_PACKET_SIZE

LogOpenFile - Remarks

The library LIBLFAPI.A must be linked with object files that use LogOpenFile

LogOpenFile - Related Functions

- [LogCloseFile](#)
- [LogReadEntry](#)

LogOpenFile - Example Code

The following example verifies that the default file (1) is opened.

```
#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>
#define ERROR_LOG_FILE_ID 1

{
    APIRET rc;                                /* return code */
    ULONG service;
    UniChar filename[256];
    ULONG filename_length;
    ULONG log_file_ID;
    LOFREQUEST open_file_packet;

    service = ERROR_LOGGING_SERVICE;

    /* Construct the LogOpenFile parameter packet */
    open_file_packet.packet_size = sizeof(LOFREQUEST);
    open_file_packet.packet_revision_number = WPOS_RELEASE_1;
    log_file_ID = ERROR_LOG_FILE_ID;
    open_file_packet.log_file_ID = &log_file_ID;
    open_file_packet.filename_length = &filename_length; /*Indicates use the
                                                         Log File ID */
    open_file_packet.filename = &filename;

    rc = LogOpenFile(service,                  /* service */
                    &open_file_packet)       /* parameter packet */

    if (rc != 0)
    {
        printf("LogOpenFile error: return code = %d",rc);
        return;
    }
}
```

*

LogOpenFile - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

LogReadEntry

LogReadEntry - Syntax

LogReadEntry reads a specified log entry from a log file.

You can specify an event filter that is used to select only entries of a desired class. The format of the event filter is described by the event-filter data structure [FILTERBLOCK](#).

```
#define INCL_LOGGING
#include <os2.h>

ULONG      service;
PVOID      pReadEntry;
APIRET     rc;

rc = LogReadEntry(service, pReadEntry);
```

LogReadEntry Parameter - service

service (ULONG) - input
The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error Logging |
| 2 | Event Tracing |

All other values are reserved for future use.

LogReadEntry Parameter - pReadEntry

pReadEntry (PVOID) - in/out
A pointer to the LogReadEntry parameter packet.

For Error Logging, this is a pointer to a [LREREQUEST](#) structure.

You provide an entry-key data structure. Entry keys are generated by this function and by [LogWaitEvent](#). Log file searching typically begins at the entry that follows the one specified within the event key. The event key is updated as new entries are read.

If no event filter is provided, LogReadEntry will read the entry that is pointed to by the event key. You also have the option of starting a search at the logical beginning of the Log File.

For Event Tracing, this is a pointer to a [LREREQUESTTRACE](#) structure.

You provide an entry ID. Log File searching typically begins at the entry that follows the one that is specified by the event ID. The event ID is updated as entries are read.

If no event filter is provided, LogReadEntry will read the entry that is pointed to by the entry ID. You also have the option of starting a search at the logical beginning of the log file.

LogReadEntry Return Value - rc

rc (APIRET) - returns
Return code.

LogReadEntry returns the following values:

- NO_ERROR
- ERROR_LF_BUF_TOO_SMALL
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_DATA_PAINTER
- INVALID_LF_PARM_PACKET_PTR
- ERROR_LF_AT_END_OF_LOG
- INVALID_LF_FLAG
- ERROR_LF_ENTRY_KEY
- ERROR_LF_INVALID_PACKET_SIZE
- INVALID_LF_FLAG

LogReadEntry - Parameters

service (ULONG) - input
The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error Logging |
| 2 | Event Tracing |

All other values are reserved for future use.

pReadEntry (PVOID) - in/out
A pointer to the LogReadEntry parameter packet.

For Error Logging, this is a pointer to a [LRREQUEST](#) structure.

You provide an entry-key data structure. Entry keys are generated by this function and by [LogWaitEvent](#). Log file searching typically begins at the entry that follows the one specified within the event key. The event key is updated as new entries are read.

If no event filter is provided, LogReadEntry will read the entry that is pointed to by the event key. You also have the option of starting a search at the logical beginning of the Log File.

For Event Tracing, this is a pointer to a [LRREQUESTTRACE](#) structure.

You provide an entry ID. Log File searching typically begins at the entry that follows the one that is specified by the event ID. The event ID is updated as entries are read.

If no event filter is provided, LogReadEntry will read the entry that is pointed to by the entry ID. You also have the option of starting a search at the logical beginning of the log file.

rc (APIRET) - returns
Return code.

LogReadEntry returns the following values:

- NO_ERROR
- ERROR_LF_BUF_TOO_SMALL
- ERROR_LF_INVALID_SERVICE
- ERROR_LF_GENERAL_FAILURE
- INVALID_DATA_POINTER
- INVALID_LF_LOG_FILE_ID
- INVALID_LF_PACKET_REVISION_NUMBER

- INVALID_DATA_PAINTER
- INVALID_LF_PARM_PACKET_PTR
- ERROR_LF_AT_END_OF_LOG
- INVALID_LF_FLAG
- ERROR_LF_ENTRY_KEY
- ERROR_LF_INVALID_PACKET_SIZE
- INVALID_LF_FLAG

LogReadEntry - Remarks

The library LIBLFAPI.A must be linked with object files that use LogReadEntry

LogReadEntry - Related Functions

- [LogAppendEntries](#)
 - [LogFormatEntry](#)
 - [LogOpenEventNotification](#)
 - [LogWaitEvent](#)
-

LogReadEntry - Example Code

The following example reads an entry from the default Error Logging log file. It searches for the first (that is, most recent) entry in the file that has a product manufacturer named "IBM" and a severity less than 4.

```
#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>
#define ERROR_LOG_FILE_ID 1
#define START_AT_FIRST_ENTRY 0 .
.
{
    ULONG service;
    ULONG severity = 4;
    ULONG entry_id = 12;
    LREREQUEST read_entry_packet;
    EVENTKEY EventKey;
    BYTE log_entry_buffer[2048];
    ULONG log_entry_buffer_length;
    SUBBLOCK subblock1, subblock2, subblock3;
    HEADERBLOCK headerblock1, headerblock2;
    FILTERBLOCK filter;

    struct UniCode_manufacturer_name {
        UniChar manufacturer_name[4] = L"IBM";
    } manufacturer_name;

    service = ERROR_LOGGING_SERVICE;

    /* Construct an event notification filter with 2 header blocks.      */
    /* The first header block points to a single subblock.              */
    /* The second header block points to a chain of two subblocks.      */
    filter.packet_size = sizeof(FILTERBLOCK);
    filter.packet_revision_number = WPOS_RELEASE_1;
    filter.header_block = &headerblock1;
    filter.header_blocks = 2;

    /*-----construct headerblock1-----*/
    headerblock1.pSubblock = &subblock1;
```



```

headerblock1.subblocks_in_block = 1;
headerblock1.pNextBlock = &headerblock2;

/*construct subblock1 of headerblock1*/
subblock1.entry_attribute_ID = LOG_ERROR_DMI_VENDOR_TAG;
subblock1.comparison_operator = LOG_ERROR_EQUAL;
subblock1.comparison_data_ptr = &manufacturer_name;
subblock1.next_subblock = NULL;

/*-----construct headerblock2-----*/
headerblock1.pSubblock = &subblock2;
headerblock1.subblocks_in_block = 2;
headerblock1.pNextBlock = NULL;

/*construct subblock2 of headerblock2*/
subblock1.entry_attribute_ID = LOG_ERROR_SEVERITY;
subblock1.comparison_operator = LOG_ERROR_LESS_THAN;
subblock1.comparison_data_ptr = severity;
subblock1.next_subblock = &subblock3;

/*construct subblock3 of headerblock2*/
subblock1.entry_attribute_ID = LOG_ERROR_ENTRY_ID;
subblock1.comparison_operator = LOG_ERROR_GREATER_THAN;
subblock1.comparison_data_ptr = entry_id;
subblock1.next_subblock = null;

/* Construct the LogReadEntry parameter packet. */
read_entry_packet.packet_size = sizeof(LRREQUEST);
read_entry_packet.packet_revision_number = WPOS_RELEASE_1;
read_entry_packet.log_file_ID = ERROR_LOG_FILE_ID;
read_entry_packet.flags = START_AT_FIRST_ENTRY;
read_entry_packet.pEventKey = &EventKey;
read_entry_packet.pFilter = &filter;
log_entry_buffer_length = sizeof(log_entry_buffer);
read_entry_packet.pLogEntryBufferLength = &log_entry_buffer_length;
read_entry_packet.LogEntryBuffer = &log_entry_buffer;

rc = LogReadEntry(service, /*service*/
                  &read_entry_packet); /*parameter packet*/

if (rc != 0)
{
    printf("LogReadEntry error: return code = %d",rc);
    return;
}

```

LogReadEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

LogWaitEvent

LogWaitEvent - Syntax

LogWaitEvent waits for event-notification information for which the process registered. You receive the event-key data structure and log entry data from the Logging Service.

For details on event-notification filters, see [LogOpenEventNotification](#).

```
#define INCL_LOGGING
#include <os2.h>

ULONG      service;
PVOID      pWaitEvent;
APIRET     rc;

rc = LogWaitEvent(service, pWaitEvent);
```

LogWaitEvent Parameter - service

service (ULONG) - input
The class of Logging Service:

- | | |
|---|---------------|
| 1 | Error Logging |
| All other values are reserved for future use. | |

LogWaitEvent Parameter - pWaitEvent

pWaitEvent (PVOID) - in/out
A pointer to the LogWaitEvent parameter packet.

For Error Logging, this is a pointer to a [LWREQUEST](#) structure.

LogWaitEvent Return Value - rc

rc (APIRET) - returns
Return code.

LogWaitEvent returns the following values:

- NO_ERROR
- ERROR_LF_BUF_TOO_SMALL
- ERROR_LF_INVALID_SERVICE
- INVALID_DATA_POINTER
- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_LF_FILENAME_LENGTH
- INVALID_LF_FILENAME_PTR
- INVALID_LF_PARM_PACKET_PTR
- INVALID_LOGRECORD_BUFFER_PTR
- INVALID_LF_FLAG
- RAS_INVALID_LOG_NOTIFY_ID
- RAS_INVALID_PACKET_SIZE
- ERROR_LF_TIMEOUT
- RAS_INVALID_EVENTKEY_PTR

- RAS_INVALID_PATHLEN_PTR
 - RAS_INVALID_BUFLLEN_PTR
-

LogWaitEvent - Parameters

service (ULONG) - input

The class of Logging Service:

1 Error Logging

All other values are reserved for future use.

pWaitEvent (PVOID) - in/out

A pointer to the LogWaitEvent parameter packet.

For Error Logging, this is a pointer to a [LWREQUEST](#) structure.

rc (APIRET) - returns

Return code.

LogWaitEvent returns the following values:

- NO_ERROR
 - ERROR_LF_BUF_TOO_SMALL
 - ERROR_LF_INVALID_SERVICE
 - INVALID_DATA_POINTER
 - INVALID_LF_PACKET_REVISION_NUMBER
 - INVALID_LF_FILENAME_LENGTH
 - INVALID_LF_FILENAME_PTR
 - INVALID_LF_PARM_PACKET_PTR
 - INVALID_LOGRECORD_BUFFER_PTR
 - INVALID_LF_FLAG
 - RAS_INVALID_LOG_NOTIFY_ID
 - RAS_INVALID_PACKET_SIZE
 - ERROR_LF_TIMEOUT
 - RAS_INVALID_EVENTKEY_PTR
 - RAS_INVALID_PATHLEN_PTR
 - RAS_INVALID_BUFLLEN_PTR
-

LogWaitEvent - Remarks

The library LIBLFAPI.A must be linked with object files that use LogWaitEvent

LogWaitEvent - Related Functions

- [LogOpenEventNotification](#)
 - [LogCloseEventNotification](#)
 - [LogChangeEventFilter](#)
 - [LogReadEntry](#)
-

LogWaitEvent - Example Code

The following example waits for an event notification from the Error Logging service. The call will wait indefinitely for an event notification.

```
#define INCL_LOGGING
#include <unichar.h>
#include <os2.h>
#include <stdio.h>
.
.
.
{
    APIRET rc;                /* return code */
    ULONG service;
    LWEREQUEST log_wait_event_packet;
    HLOGNOTIFY log_notify;
    EVENTKEY EventKey;
    BYTE log_entry_buffer[4096];
    UniChar pathname[512];
    ULONG pathname_length = sizeof(pathname);

    service = ERROR_LOGGING_SERVICE;

    /* Construct the LogChangeEventFilter parameter packet */
    log_wait_event_packet.packet_size = sizeof(LWEREQUEST);
    log_wait_event_packet.packet_revision_number = WPOS_RELEASE_1;
    log_wait_event_packet.LogNotify = log_notify;
    log_wait_event_packet.pEventKey = &EventKey;
    log_wait_event_packet.pLogEntryBuffer = &log_entry_buffer;
    log_wait_event_packet.timeout = 0;
    log_wait_event_packet.queue_flags = 0;
    log_wait_event_packet.pathname_length = &pathname_length;
    log_wait_event_packet.pathname = pathname;

    rc = LogWaitEvent(service,                /* service */
                     &log_wait_event_packet /* parameter packet */
    );
    if (rc != 0)
    {
        printf("LogWaitEvent error: return code = %d", rc);
        return;
    }
}
```

LogWaitEvent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

TraceCreateEntry

TraceCreateEntry - Syntax

TraceCreateEntry creates an event entry in the system event buffer. It is the static trace-point creation mechanism.

```
#define INCL_TRACE
#include <os2.h>

PTCEREQUEST    pTraceCreateEntry;
APIRET         rc;

rc = TraceCreateEntry(pTraceCreateEntry);
```

TraceCreateEntry Parameter - pTraceCreateEntry

pTraceCreateEntry (**PTCEREQUEST**) - input
Pointer to the TraceCreateEntry parameter packet.

TraceCreateEntry Return Value - rc

rc (**APIRET**) - returns
TraceCreateEntry returns the following values:

- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_DATA_POINTER
- INVALID_TRACE_MAJOR_CODE
- INVALID_TRACE_MINOR_CODE
- INVALID_PACKET_SIZE

TraceCreateEntry - Parameters

pTraceCreateEntry (**PTCEREQUEST**) - input
Pointer to the TraceCreateEntry parameter packet.

rc (**APIRET**) - returns
TraceCreateEntry returns the following values:

- INVALID_LF_PACKET_REVISION_NUMBER
- INVALID_DATA_POINTER
- INVALID_TRACE_MAJOR_CODE
- INVALID_TRACE_MINOR_CODE
- INVALID_PACKET_SIZE

TraceCreateEntry - Remarks

Event trace records contain information that describes the occurrence of software events. They can be used both as service aids and in performance monitoring.

The library LIBTRACE.A must be linked with object files that use TraceCreateEntry.

TraceCreateEntry - Example Code

The following example adds an event trace entry to the system trace buffer. For this example, the trace entry will contain the contents of two internal program variables.

```
#define INCL_DOSPROCESS

#include <stdio.h>          /* C library for standard I/O      */
#include <stdlib.h>         /* C library of standard routines  */
#include <string.h>         /* C library for string operations  */
#include <os2.h>            /* OS/2 Dos api calls              */
#include <trace.h>         /* Trace public API data structures */

#define HKWD_TEST          43
#define hkwd_test_entry    0001

struct {
    ULONG var1;
    USHORT var2;
} trace_data;

TCEREQUEST trace_create_entry_packet;

VOID main(VOID)
{
    APIRET rc = NO_ERROR;

    /******
    /* Set up the TraceCreateEntry parameter packet
    /******
    trace_create_entry_packet.packet_size      = sizeof(TCEREQUEST);
    trace_create_entry_packet.packet_revision_number = WPOS_RELEASE_1;
    trace_create_entry_packet.major_event_code   = HKWD_TEST;
    trace_create_entry_packet.minor_event_code   = hkwd_test_entry;
    trace_create_entry_packet.event_data_length  = sizeof(trace_data);
    trace_create_entry_packet.event_data        = (PVOID)&trace_data;

    /******
    /* Place tracepoint data in the tracepoint data buffer
    /******
    trace_data.var1 = UINT_MAX;
    trace_data.var2 = 1;

    rc = TraceCreateEntry(&trace_create_entry_packet);
    if (rc != NO_ERROR) {
        printf("TraceCreateEntry RC(%d)\n", rc);
    }
}
```

TraceCreateEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

Problem Determination Data Types

This section describes the following data types that are used with the Problem Determination functions:

- CONFIGPARMS
- DISABLEPRODUCT
- DMIDATA
- DUMPDATAVAR
- DUMPUSERDATA
- EVENTKEY
- FFSTPARMS
- FILTERBLOCK
- HEADERBLOCK
- LCEFREQUEST
- LGENREQUEST
- LCFREQUEST
- LCFREQUESTTRACE
- LFERREQUEST
- LOENREQUEST
- LOFREQUEST
- LOFREQUESTTRACE
- LRERREQUEST
- LRERREQUESTTRACE
- LWERREQUEST
- MODINFO
- MSGINSDATA
- MSGINSTXT
- PRODUCTDATA
- PRODUCTINFO
- SUBBLOCK
- TCERREQUEST

CONFIGPARMS

```
typedef struct _CONFIGPARMS {
    ULONG          packet_size;
    ULONG          packet_revision_number;
    ULONG          dump_file_wrap;
    ULONG          dump_file_directory_name_length;
    UniChar        *dump_file_directory_name;
    ULONG          no_of_probe_disabled_products;
    PDISABLEPRODUCT pDisableProduct;
    ULONG          message_pop_up;
} CONFIGPARMS;

typedef CONFIGPARMS *PCONFIGPARMS;
```

CONFIGPARMS Field - packet_size

packet_size (ULONG)
Size, in bytes, of this packet.

CONFIGPARMS Field - packet_revision_number

packet_revision_number (ULONG)

Revision number for this packet. Should be set to:

CONFIGPARMS_WPOS_REVISION_NUMBER_1

CONFIGPARMS Field - dump_file_wrap

dump_file_wrap (ULONG)

Indicates whether the dumps captured by FFST are to be wrapped or not.

If dump wrap is set to on then dumps will be wrapped when the specified maximum size is reached. If dump wrap is set to off when the specified maximum size is reached then FFST will not create any new dump files.

May be one of the following values:

0	Set dump wrap to off
1	Set dump wrap to on (system default)

CONFIGPARMS Field - dump_file_directory_name_length

dump_file_directory_name_length (ULONG)

Length, in bytes, of the dump directory name.

CONFIGPARMS Field - dump_file_directory_name

dump_file_directory_name (UniChar *)

Pointer to the fully-qualified dump directory name.

CONFIGPARMS Field - no_of_probe_disabled_products

no_of_probe_disabled_products (ULONG)

Number of products whose FFSTProbes are disabled.

CONFIGPARMS Field - pDisableProduct

pDisableProduct ([PDISABLEPRODUCT](#))

Pointer to the structure that identifies the product data of the disabled probe. This parameter can be ignored when

no_of_probe_disabled_products is 0 or 'FFFFFFFF'.

CONFIGPARMS Field - message_pop_up

message_pop_up (ULONG)
Reserved parameter.

DISABLEPRODUCT

```
typedef struct _DISABLEPRODUCT {  
    UniChar    *DMI_vendor_tag;  
    UniChar    *DMI_tag;  
    UniChar    *DMI_revision;  
} DISABLEPRODUCT;  
  
typedef DISABLEPRODUCT *PDISABLEPRODUCT;
```

DISABLEPRODUCT Field - DMI_vendor_tag

DMI_vendor_tag (UniChar *)
Pointer to the short product manufacturer name that was logged in the DMI database.

DISABLEPRODUCT Field - DMI_tag

DMI_tag (UniChar *)
Pointer to the short product name that was logged in the DMI database.

DISABLEPRODUCT Field - DMI_revision

DMI_revision (UniChar *)
Pointer to the product revision information that was logged in the DMI database.

DMIDATA

```
typedef struct _DMIDATA {
    ULONG        packet_size;
    ULONG        packet_revision_number;
    UniChar      *DMI_product_ID;
    UniChar      *DMI_modification_level;
    UniChar      *DMI_fix_level;
    ULONG        template_filename_length;
    UniChar      *template_filename;
} DMIDATA;

typedef DMIDATA *PDMIDATA;
```

DMIDATA Field - packet_size

packet_size (ULONG)
Length, in bytes, of this packet.

DMIDATA Field - packet_revision_number

packet_revision_number (ULONG)
Revision number for this packet. Should be set to:

DMIDATA_REVISION_NUMBER_1

DMIDATA Field - DMI_product_ID

DMI_product_ID (UniChar *)
Pointer to the product ID.

DMIDATA Field - DMI_modification_level

DMI_modification_level (UniChar *)
Pointer to the product modification level.

DMIDATA Field - DMI_fix_level

DMI_fix_level (UniChar *)
Pointer to the product fix level.

DMIDATA Field - template_filename_length

template_filename_length (ULONG)
Length of the template repository file name pointed to by *template_filename* .

DMIDATA Field - template_filename

template_filename (UniChar *)
Pointer to the fully-qualified path of the template repository file.

DUMPDATAVAR

```
typedef struct _DUMPDATAVAR {  
    ULONG      var_n_length;  
    PVOID      var_n;  
} DUMPDATAVAR;  
  
typedef DUMPDATAVAR *PDUMPDATAVAR;
```

DUMPDATAVAR Field - var_n_length

var_n_length (ULONG)
Length, in bytes, of the data structure pointed to by *var_n* .

DUMPDATAVAR Field - var_n

var_n (PVOID)
Pointer to the data structure to be collected.

DUMPUSEADATA

```
typedef struct _DUMPUSERDATA {
    ULONG      no_of_variables;
    DUMPDATAVAR DumpDataVar[MAX_USER_DUMPS];
} DUMPUSERDATA;

typedef DUMPUSERDATA *PDUMPUSERDATA;
```

DUMPUSERDATA Field - no_of_variables

no_of_variables (ULONG)

Number of data/structures to be collected by the probe. Maximum number of variables that can be collected is 30.

DUMPUSERDATA Field - DumpDataVar[MAX_USER_DUMPS]

DumpDataVar[MAX_USER_DUMPS] (DUMPDATAVAR)

Dump data variables repeated *no_of_variables* times.

EVENTKEY

Event key data structure.

```
typedef struct _EVENTKEY {
    ULONG      location;
    ULONG      entry_ID;
} EVENTKEY;

typedef EVENTKEY *PEVENTKEY;
```

EVENTKEY Field - location

location (ULONG)

Location within the log file.

EVENTKEY Field - entry_ID

entry_ID (ULONG)

Entry ID of the record.

FFSTPARMS

```
typedef struct _FFSTPARMS {
    ULONG          packet_size;
    ULONG          packet_revision_number;
    UniChar        *module_name;
    ULONG          probe_ID;
    ULONG          severity;
    ULONG          template_record_ID;
    PMSGINSDATA    pMsgInsData;
    ULONG          probe_flags;
    PDUMPUSEDATA   pDumpUserData;
    ULONG          log_user_data_length;
    PVOID          log_user_data;
} FFSTPARMS;

typedef FFSTPARMS *PFFSTPARMS;
```

FFSTPARMS Field - packet_size

packet_size (ULONG)
Length, in bytes, of the packet.

FFSTPARMS Field - packet_revision_number

packet_revision_number (ULONG)
Revision number for this packet. Should be set to:

FFSTPARMS_WPOS_REVISION_NUMBER_1

FFSTPARMS Field - module_name

module_name (UniChar *)
Address of the module name.

FFSTPARMS Field - probe_ID

probe_ID (ULONG)
Unique identifier of the detection point within the module_name.

FFSTPARMS Field - severity

severity (ULONG)
The severity of this error. Valid values are 1 (highest) through 6 (lowest):

- | | |
|---|--|
| 1 | SEVERITY1
<i>Critical:</i> Condition from which there is no recovery. |
| 2 | SEVERITY2
<i>Major:</i> Condition signifying that the loss of availability of a device or subproduct is imminent, or the performance of the device or subproduct has degraded to below an acceptable level. |
| 3 | SEVERITY3
<i>Minor:</i> Condition that was recovered from after a number of attempts or a nonservice-affecting fault has occurred that should be corrected before a more serious error occurs. |
| 4 | SEVERITY4
<i>Warning:</i> A potential error has been detected before any significant effects have been felt. |
| 5 | SEVERITY5
<i>Indeterminate:</i> Condition where it is not possible to determine the severity of the error. |
| 6 | SEVERITY6
<i>Information</i> |

For severity 1, 2 and 3, FFST will request trace information, if system trace is 'ON'. For severity 4, 5 and 6, FFST will not request trace information, unless the end user specifically requests it through a probe control table entry.

FFSTPARMS Field - template_record_ID

template_record_ID (ULONG)
Identifier to specify the Log Entry Format Template that is be used by SYSLOG utility.

FFSTPARMS Field - pMsgInsData

pMsgInsData ([PMSGINSDATA](#))
A pointer to message insert data. NULL indicates that there are no insert texts for the message.

FFSTPARMS Field - probe_flags

probe_flags (ULONG)
Indicates what type of system information is needed and where to keep the information that is collected. If PSTAT or Process Environment are requested, FFST will create a dump file and will keep this information in the dump file. The user can specify a combination of these flags. As an example, if it is desired to capture both PSTAT and Process Environment information, then the value specified will be 3.

If the user sets any of the probe flags listed below, FFST will create an FFST dump file where the information will be stored. The exceptions are Process Dump (PROCESS_DUMP_FLAG) and System Trace (CAPTURE_TRACE). These processes will have their

own files that FFST will maintain. Trace information will be automatically generated for severity 1, 2, and 3 probes. For severity 4, 5 and 6 probes, trace will not be collected.

PSTAT_FLAG (0x01) Capture PSTAT information.
PROC_ENV_FLAG (0x02) Capture process environment information.

FFSTPARMS Field - pDumpUserData

pDumpUserData (PDUMPUSERDATA)

Pointer to user_data containing various storage areas. Maximum storage areas that can be specified are 30. NULL indicates that there are no user data items for this probe call. The items specified will be stored in a dump file created by FFST. Each individual data area can have a maximum size of 32K.

FFSTPARMS Field - log_user_data_length

log_user_data_length (ULONG)

The length in bytes of *log_user_data*. A length of 0 means there is no user data.

FFSTPARMS Field - log_user_data

log_user_data (PVOID)

Pointer to data to be placed in the Error Log File. The data can be formatted through the template for display. If no user data is desired, then the pointer must be null. This data will be stored in the Error Log File as part of the log entry.

FILTERBLOCK

Filter block structure.

```
typedef struct _FILTERBLOCK {  
    ULONG      packet_size;  
    ULONG      packet_revision_number;  
    PHEADERBLOCK header_block;  
} FILTERBLOCK;  
  
typedef FILTERBLOCK *PFILTERBLOCK;
```

FILTERBLOCK Field - packet_size

packet_size (ULONG)

The size, in bytes, of this packet.

FILTERBLOCK Field - packet_revision_number

packet_revision_number (ULONG)

A long integer value that indicates the revision level of the parameter packet.

2	WPOS_RELEASE_1
	Indicates OS/2 Warp (PowerPC Edition) Release 1 level parameter packet.

FILTERBLOCK Field - header_block

header_block (PHEADERBLOCK)

Pointer to the first header block.

HEADERBLOCK

Header block for an event notification filter structure.

```
typedef struct _HEADERBLOCK {
    PSUBBLOCK      pSubblock;
    struct _HEADERBLOCK *pNextBlock;
} HEADERBLOCK;

typedef HEADERBLOCK *PHEADERBLOCK;
```

HEADERBLOCK Field - pSubblock

pSubblock (PSUBBLOCK)

Pointer to the first sub-block.

HEADERBLOCK Field - pNextBlock

pNextBlock (struct _HEADERBLOCK *)

Pointer to the next header block in the chain. A NULL indicates the end of the chain.

LCEFREQUEST

LogChangeEventFilter parameter packet.

```
typedef struct _LCEFREQUEST {  
    ULONG          packet_size;  
    ULONG          packet_revision_number;  
    ULONG          purge_flags;  
    ULONG          LogNotify;  
    PFILTERBLOCK   pFilter;  
} LCEFREQUEST;  
  
typedef LCEFREQUEST *PLCEFREQUEST;
```

LCEFREQUEST Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

LCEFREQUEST Field - packet_revision_number

packet_revision_number (ULONG)
A long integer value that indicates the revision level of the parameter packet.

2 Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

LCEFREQUEST Field - purge_flags

purge_flags (ULONG)
Purge event notification queue.

LCEFREQUEST Field - LogNotify

LogNotify (ULONG)
Handle of event notification queue.

LCEFREQUEST Field - pFilter

pFilter (PFILTERBLOCK)
Pointer to the event filter data.

LCENREQUEST

LogCloseEventNotification parameter packet.

```
typedef struct _LCENREQUEST {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    ULONG      LogNotify;
} LCENREQUEST;

typedef LCENREQUEST *PLCENREQUEST;
```

LCENREQUEST Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

LCENREQUEST Field - packet_revision_number

packet_revision_number (ULONG)
A long integer value that indicates the revision level of the parameter packet.

2	Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.
---	---

LCENREQUEST Field - LogNotify

LogNotify (ULONG)
Handle of event notification queue.

LCFREQUEST

LogCloseFile parameter packet.

```
typedef struct _LCFREQUEST {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    ULONG      purge_flags;
    ULONG      LogNotify;
    PFILTERBLOCK pFilter;
} LCFREQUEST;
```

```
typedef LCFREQUEST *PLCFREQUEST;
```

LCFREQUEST Field - packet_size

packet_size (ULONG)

The size, in bytes, of this packet.

LCFREQUEST Field - packet_revision_number

packet_revision_number (ULONG)

A long integer value that indicates the revision level of the parameter packet.

2 Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

LCFREQUEST Field - purge_flags

purge_flags (ULONG)

Purge event notification queue.

LCFREQUEST Field - LogNotify

LogNotify (ULONG)

Handle of event notification queue.

LCFREQUEST Field - pFilter

pFilter ([PFILTERBLOCK](#))

Pointer to the event filter data.

LCFREQUESTTRACE

Trace version of LogCloseFile parameter packet.

```

typedef struct _LCFREQUESTTRACE {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    HFILE      FileHandle;          /* File handle. */
} LCFREQUESTTRACE;

typedef LCFREQUESTTRACE *PLCFREQUESTTRACE;

```

LCFREQUESTTRACE Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

LCFREQUESTTRACE Field - packet_revision_number

packet_revision_number (ULONG)
A long integer value that indicates the revision level of the parameter packet.

2 Indicates WPOS Release 1-level parameter packet.

LCFREQUESTTRACE Field - FileHandle

FileHandle (HFILE)
File handle.

LFEREQUEST

LogFormatEntry parameter packet.

```

typedef struct _LFEREQUEST {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    PVOID      log_entry_buffer;
    PVOID      locale_object;
    PULONG     number_of_detail_records;
    ULONG      flags;                /* Flags. */
    PULONG     string_buffer_length;
    UniChar    *string_buffer;
} LFEREQUEST;

typedef LFEREQUEST *PLFEREQUEST;

```

LFEREQUEST Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

LFEREREQUEST Field - packet_revision_number

packet_revision_number (ULONG)
A long integer value that indicates the revision level of the parameter packet.

2 Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

LFEREREQUEST Field - log_entry_buffer

log_entry_buffer (PVOID)
Pointer to a Log Entry record.

LFEREREQUEST Field - locale_object

locale_object (PVOID)

LFEREREQUEST Field - number_of_detail_records

number_of_detail_records (PULONG)

LFEREREQUEST Field - flags

flags (ULONG)
Flags.

LFEREREQUEST Field - string_buffer_length

string_buffer_length (PULONG)

On input, a pointer to the caller's string. On output, size of string_buffer.

LFEREQUEST Field - string_buffer

string_buffer (UniChar *)

Pointer to the output buffer.

LFEREQUESTTRACE

LogFormatEntry event tracing parameter packet.

```
typedef struct _LFEREQUESTTRACE {
    ULONG         packet_size;
    ULONG         packet_revision_number;
    PVOID         log_entry_buffer;
    PVOID         locale_object;
    PULONG        number_of_detail_records;
    ULONG         flags; /* flags. */
    PULONG        string_buffer_length;
    UniChar       *string_buffer;
    ULONG         pathname_table_size;
    PVOID         pathname_table;
} LFEREQUESTTRACE;
```

```
typedef LFEREQUESTTRACE *PLFEREQUESTTRACE;
```

LFEREQUESTTRACE Field - packet_size

packet_size (ULONG)

The size, in bytes, of this packet.

LFEREQUESTTRACE Field - packet_revision_number

packet_revision_number (ULONG)

A long integer value that indicates the revision level of the parameter packet.

2 Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

LFEREQUESTTRACE Field - log_entry_buffer

log_entry_buffer (PVOID)
Pointer to a Log Entry record.

LFEREQUESTTRACE Field - locale_object

locale_object (PVOID)

LFEREQUESTTRACE Field - number_of_detail_records

number_of_detail_records (PULONG)

LFEREQUESTTRACE Field - flags

flags (ULONG)
flags.

LFEREQUESTTRACE Field - string_buffer_length

string_buffer_length (PULONG)
On input, a pointer to the caller's string. On output, size of string_buffer.

LFEREQUESTTRACE Field - string_buffer

string_buffer (UniChar *)
Pointer to the output buffer.

LFEREQUESTTRACE Field - pathname_table_size

pathname_table_size (ULONG)

Size of pathname_table.

LFERESTTTRACE Field - pathname_table

pathname_table (PVOID)
Pointer to pathname table.

LOENREQUEST

LogOpenEventNotification parameter packet.

```
typedef struct _LOENREQUEST {  
    ULONG          packet_size;  
    ULONG          packet_revision_number;  
    ULONG          log_file_ID;  
    ULONG          read_flags;  
    PULONG         pLogNotify;  
    PFILTERBLOCK   pFilter;  
} LOENREQUEST;
```

```
typedef LOENREQUEST *PLOENREQUEST;
```

LOENREQUEST Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

LOENREQUEST Field - packet_revision_number

packet_revision_number (ULONG)
A long integer value that indicates the revision level of the parameter packet.

2	WPOS_RELEASE_1 Indicates OS/2 Warp (PowerPC Edition) Release 1 level parameter packet.
---	---

LOENREQUEST Field - log_file_ID

log_file_ID (ULONG)
ID for this log file. For Release 1, this is restricted to the current log file, which has and ID of 1.

LOENREQUEST Field - read_flags

read_flags (ULONG)

Flag values used to control the notification.

0	Return the record
1	Do not return the record.

LOENREQUEST Field - pLogNotify

pLogNotify (PULONG)

Pointer to the handle of the event queue.

LOENREQUEST Field - pFilter

pFilter (PFILTERBLOCK)

Pointer to the event filter data structure. A NULL indicates that no initial filter is specified.

LOFREQUEST

LogOpenFilter parameter packet.

```
typedef struct _LOFREQUEST {  
    ULONG        packet_size;  
    ULONG        packet_revision_number;  
    PULONG       log_file_ID;  
    PULONG       filename_length;  
    UniChar      *filename;  
} LOFREQUEST;
```

```
typedef LOFREQUEST *PLOFREQUEST;
```

LOFREQUEST Field - packet_size

packet_size (ULONG)

The size, in bytes, of this packet.

LOFREQUEST Field - packet_revision_number

packet_revision_number (ULONG)

A long integer value that indicates the revision level of the parameter packet.

2 Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

LOFREQUEST Field - log_file_ID

log_file_ID (PULONG)

ID of the file to be opened.

LOFREQUEST Field - filename_length

filename_length (PULONG)

Length of filename.

LOFREQUEST Field - filename

filename (UniChar *)

Pointer to the file name.

LOFREQUESTTRACE

Trace version of LogOpenFile parameter packet.

```
typedef struct _LOFREQUESTTRACE {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    PULONG     filename_length;
    UniChar    *filename;
    HFILE      FileHandle;
} LOFREQUESTTRACE;

typedef LOFREQUESTTRACE *PLOFREQUESTTRACE;
```

LOFREQUESTTRACE Field - packet_size

packet_size (ULONG)

The size, in bytes, of this packet.

LOFREQUESTTRACE Field - packet_revision_number

packet_revision_number (ULONG)

A long integer value that indicates the revision level of the parameter packet.

2 Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

LOFREQUESTTRACE Field - filename_length

filename_length (PULONG)

Length of filename.

LOFREQUESTTRACE Field - filename

filename (UniChar *)

Pointer to the file name.

LOFREQUESTTRACE Field - FileHandle

FileHandle (HFILE)

The file handle.

LRREQUEST

LogReadFile parameter packet.

```
typedef struct _LRREQUEST {
    ULONG          packet_size;
    ULONG          packet_revision_number;
    ULONG          log_file_ID;
    ULONG          flags;
    PEVENTKEY      pEventKey;
    PFILTERBLOCK    pFilter;
    PULONG          pLogEntryBufferLength;
    PVOID           pLogEntryBuffer;
} LRREQUEST;
```

```
typedef LRREQUEST *PLRREQUEST;
```

LRREQUEST Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

LRREQUEST Field - packet_revision_number

packet_revision_number (ULONG)
A long integer value that indicates the revision level of the parameter packet.

2	Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.
---	---

LRREQUEST Field - log_file_ID

log_file_ID (ULONG)
ID of the log file to be used.

LRREQUEST Field - flags

flags (ULONG)
Relative position to begin file search.

LRREQUEST Field - pEventKey

pEventKey ([PEVENTKEY](#))
Pointer to the event key data structure.

LRREQUEST Field - pFilter

pFilter ([PFILTERBLOCK](#))
Pointer to the event filter to use for the search.

LREREREQUEST Field - pLogEntryBufferLength

pLogEntryBufferLength (PULONG)

On input, length of caller's buffer. On output, size of LogEntryBuffer.

LREREREQUEST Field - pLogEntryBuffer

pLogEntryBuffer (PVOID)

Pointer to the buffer containing the log record.

LREREREQUESTTRACE

Trace version of LogReadFile parameter packet.

```
typedef struct _LREREREQUESTTRACE {
    ULONG          packet_size;
    ULONG          packet_revision_number;
    HFILE          FileHandle;
    ULONG          flags;                      /* Single or multiple read. */
    ULONG          entry_ID;                  /* Entry ID of where to begin reading. */
    PFILTERBLOCK   pFilter;
    PULONG         pLogEntryBufferLength;
    PVOID          pLogEntryBuffer;
} LREREREQUESTTRACE;

typedef LREREREQUESTTRACE *PLREREREQUESTTRACE;
```

LREREREQUESTTRACE Field - packet_size

packet_size (ULONG)

The size, in bytes, of this packet.

LREREREQUESTTRACE Field - packet_revision_number

packet_revision_number (ULONG)

A long integer value that indicates the revision level of the parameter packet.

2 Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

LREREREQUESTTRACE Field - FileHandle

FileHandle (HFILE)
File handle.

LRREQUESTTRACE Field - flags

flags (ULONG)

Single or multiple read.

May be one of the following values:

- | | |
|---|--|
| 0 | TRACE_READ_SINGLE_RECORD
Return a single entry. |
| 1 | TRACE_READ_MULTIPLE_RECORDS
Return multiple entries (as many complete records that will fit in the caller's buffer or until end-of-file). |
-

LRREQUESTTRACE Field - entry_ID

entry_ID (ULONG)

Entry ID of where to begin reading.

On output, this value is the entry ID prior to the last entry ID returned.

On input, may be one of the following values:

- | | |
|---------------------------------|------------------------------------|
| TRACE_NEWEST_ENTRY (0xFFFFFFFF) | Start at the most recent entry. |
| TRACE_OLDEST_ENTRY (0x00000001) | Start at the the oldest entry. |
| n | Start with the entry ID specified. |
-

LRREQUESTTRACE Field - pFilter

pFilter (PFILTERBLOCK)

Pointer to the event filter to use for the search.

LRREQUESTTRACE Field - pLogEntryBufferLength

pLogEntryBufferLength (PULONG)

On input, the length of the caller's buffer. On output, the size of *pLogEntryBuffer* . If the caller's buffer is too small to hold a single entry, *pLogEntryBufferLength* will be set to the required size to hold a single entry and no data will be returned.

LRREQUESTTRACE Field - pLogEntryBuffer

pLogEntryBuffer (PVOID)
Pointer to the buffer containing the log record.

LWEREQUEST

LogWaitEvent parameter packet.

```
typedef struct _LWEREQUEST {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    ULONG      LogNotify;
    PEVENTKEY  pEventKey;
    PULONG     log_entry_buffer_length;
    PVOID      pLogEntryBuffer;
    ULONG      timeout;
    ULONG      queue_flags;
    ULONG      pathname_length;
    UniChar    *pathname;
} LWEREQUEST;

typedef LWEREQUEST *PLWEREQUEST;
```

LWEREQUEST Field - packet_size

packet_size (ULONG)
The size, in bytes, of this packet.

LWEREQUEST Field - packet_revision_number

packet_revision_number (ULONG)
A long integer value that indicates the revision level of the parameter packet.

2	WPOS_RELEASE_1
	Indicates OS/2 Warp (PowerPC Edition) Release 1 level parameter packet.

LWEREQUEST Field - LogNotify

LogNotify (ULONG)
Event notification ID.

LWEREQUEST Field - pEventKey

pEventKey ([PEVENTKEY](#))

Pointer to the event key data structure.

This data structure can subsequently be passed to [LogReadEntry](#) so that the event consumer can read the log file entry that is associated with this event notification.

LWREQUEST Field - log_entry_buffer_length

log_entry_buffer_length (PULONG)

The size, in bytes, of the provided buffer pointed to by *pLogEntryBuffer* .

On input, this is the length of the caller's provided buffer. On output, this is the total number of bytes placed in the provided buffer. If the caller's buffer is too small, then this will be set to the required size and no data will be written to the buffer.

LWREQUEST Field - pLogEntryBuffer

pLogEntryBuffer (PVOID)

Pointer to the buffer that is to receive the selected error log record.

LWREQUEST Field - timeout

timeout (ULONG)

Milliseconds to wait before timing out. A value of 0 indicates to wait indefinitely.

LWREQUEST Field - queue_flags

queue_flags (ULONG)

Flags that indicate how the queue will be handled if the buffer provided by the user is not large enough to hold the record.

May be one of the following values:

- | | |
|---|---|
| 0 | Do not remove record from queue. |
| 1 | Remove record from queue. (If the record is still required, it can be retrieved from the log file by using the EVENTKEY structure and LogReadEntry .) |

LWREQUEST Field - pathname_length

pathname_length (ULONG)
Length of the pathname.

On input, this is the length of the caller's provided *pathname* . On output, this is the total number of bytes placed in *pathname* . If the caller's *pathname* length is too small, then this will be set to the required length and no data will be written to *pathname* .

LWEREQUEST Field - pathname

pathname (UniChar *)
Pointer to the pathname where the error log record being reported in LogEntryBuffer resides.

MODINFO

```
typedef struct _MODINFO {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    PUNICHAR    subproduct_name;
    PUNICHAR    module_name;
} MODINFO;

typedef MODINFO *PMODINFO;
```

MODINFO Field - packet_size

packet_size (ULONG)
Length, in bytes, of this packet.

MODINFO Field - packet_revision_number

packet_revision_number (ULONG)
Revision number for this packet.

MODINFO_REVISION_NUMBER_1 1

MODINFO Field - subproduct_name

subproduct_name (PUNICHAR)
Pointer to the subproduct name.

MODINFO Field - module_name

module_name (PUNICHAR)

Pointer to the module identifier name. Identifies the originator of the probe.

MSGINSDATA

```
typedef struct _MSGINSDATA {
    ULONG          no_inserts;
    MSGINSTXT      MsgInsTxt[MAX_MSG_INSERTS];
} MSGINSDATA;

typedef MSGINSDATA *PMSGINSDATA;
```

MSGINSDATA Field - no_inserts

no_inserts (ULONG)

Number of insert strings provided. Maximum allowed is 9.

MSGINSDATA Field - MsgInsTxt[MAX_MSG_INSERTS]

MsgInsTxt[MAX_MSG_INSERTS] (MSGINSTXT)

Message insert text details repeated *no_inserts* times.

MSGINSTXT

```
typedef struct _MSGINSTXT {
    ULONG          insert_number;
    UniChar        *insert_text;
} MSGINSTXT;

typedef MSGINSTXT *PMSGINSTXT;
```

MSGINSTXT Field - insert_number

insert_number (ULONG)

The insert number. This must match the %n in the message.

MSGINSTXT Field - insert_text

insert_text (UniChar *)

Address of the message text.

PRODUCTDATA

```
typedef struct _PRODUCTDATA {  
    ULONG        packet_size;  
    ULONG        packet_revision_number;  
    UniChar      *DMI_vendor_tag;  
    UniChar      *DMI_tag;  
    UniChar      *DMI_revision;  
} PRODUCTDATA;
```

```
typedef PRODUCTDATA *PPRODUCTDATA;
```

PRODUCTDATA Field - packet_size

packet_size (ULONG)

Length, in bytes, of this packet.

PRODUCTDATA Field - packet_revision_number

packet_revision_number (ULONG)

Revision number for this packet. Should be set to:

```
PRODUCTDATA_REVISION_NUMBER_1  
1
```

PRODUCTDATA Field - DMI_vendor_tag

DMI_vendor_tag (UniChar *)
Pointer to the name of the product manufacturer.

PRODUCTDATA Field - DMI_tag

DMI_tag (UniChar *)
Pointer to the product name that was logged in the DMI database.

PRODUCTDATA Field - DMI_revision

DMI_revision (UniChar *)
Pointer to the product version number.

PRODUCTINFO

```
typedef struct _PRODUCTINFO {  
    PPRODUCTDATA    pProductData;  
    PDMIDATA         pDMIData;  
} PRODUCTINFO;  
  
typedef PRODUCTINFO *PPRODUCTINFO;
```

PRODUCTINFO Field - pProductData

pProductData (PPRODUCTDATA)
Pointer to a structure that identifies the module in which the probe is inserted.

PRODUCTINFO Field - pDMIData

pDMIData (PDMIDATA)
Pointer to a structure that specifies the product information if DMI is not used. A NULL value indicates that the DMI information will be retrieved from the DMI Repository (recommended approach).

SUBBLOCK

Selection criteria sub-block structure.

```
typedef struct _SUBBLOCK {
    ULONG          entry_attribute_ID; /* The identifier of a field within a log entry. */
    ULONG          comparison_operator_ID; /* The identifier of a comparison operator. */
    ULONG          comparison_data_length; /* The length of the comparison data. */
    PVOID          comparison_data_ptr;
    struct _SUBBLOCK *next_subblock;
} SUBBLOCK;

typedef SUBBLOCK *PSUBBLOCK;
```

SUBBLOCK Field - entry_attribute_ID

entry_attribute_ID (ULONG)

The identifier of a field within a log entry.

The field within an entry that will be compared against the target data value that is pointed to by *comparison_data_ptr*.

May be one of the following values for Error Logging:

LOG_ERROR_DATE	Date type
LOG_ERROR_TIME	Time type
LOG_ERROR_ENTRY_ID	Unsigned long integer type
LOG_ERROR_RECORD_TYPE	String type
LOG_ERROR_SEVERITY	Unsigned long integer type
LOG_ERROR_PROCESS_PATHNAME	String type
LOG_ERROR_SOURCE_MODULE_NAME	String type
LOG_ERROR_PROBE_ID	Unsigned long integer type
LOG_ERROR_DMI_VENDOR_TAG	String type
LOG_ERROR_DMI_TAG	String type
LOG_ERROR_DMI_REVISION	String type
LOG_ERROR_MACHINE_TYPE	String type
LOG_ERROR_SERIAL_NUMBER	String type
LOG_ERROR_USER_DATA	String type

May be one of the following values for Event Tracing:

LOG_TRACE_TIMESTAMP	Time stamp.
LOG_TRACE_MAJOR_CODE	Unsigned long integer type
LOG_TRACE_MINOR_CODE	Unsigned long integer type
LOG_TRACEOS2_PROCESS_ID	Unsigned long integer type
LOG_TRACEOS2_THREAD_ID	Unsigned long integer type
LOG_TRACE_MICROKERNEL_TASK_ID	Unsigned long integer type
LOG_TRACE_MICROKERNEL_THREAD_ID	Unsigned long integer type
LOG_TRACE_EVENT_NUMBER	Unsigned long integer type

SUBBLOCK Field - comparison_operator_ID

comparison_operator_ID (ULONG)

The identifier of a comparison operator.

This comparison operator must be valid for the type of log entry data item that was specified by *entry_attribute_ID*.

May be one of the following values:

LOG_ERROR_EQUAL

LOG_ERROR_NOT_EQUAL

LOG_ERROR_GREATER_THAN

LOG_ERROR_GREATER_THAN_OR_EQUAL

LOG_ERROR_LESS_THAN

LOG_ERROR_LESS_THAN_OR_EQUAL

LOG_SUBSTRING_MATCH

Date, time, string, and unsigned long integer types

Date, time, string, and unsigned long integer types

Date, time, string, and unsigned long integer types

Date, time, string, and unsigned long integer types

Date, time, string, and unsigned long integer types

Date, time, string, and unsigned long integer types

String type only

SUBBLOCK Field - comparison_data_length

comparison_data_length (ULONG)

The length of the comparison data.

SUBBLOCK Field - comparison_data_ptr

comparison_data_ptr (PVOID)

Pointer to a data item that will be compared against the specified log entry attribute.

Note: The data item is expected to be in the proper format. An example is the date and time attributes, which require data in the format that is maintained in the Log File.

SUBBLOCK Field - next_subblock

next_subblock (struct _SUBBLOCK *)

Pointer to the next sub-block in the chain. A NULL indicate the end of the chain.

TCEREREQUEST

Structure for TraceCreateEntry.

```
typedef struct _TCEREREQUEST {
    ULONG      packet_size;
    ULONG      packet_revision_number;
    ULONG      major_event_code;
    ULONG      minor_event_code;
    ULONG      event_data_length;
    PVOID      event_data;
} TCEREREQUEST;
```

```
typedef TCEREREQUEST *PTCEREREQUEST;
```

TCEREREQUEST Field - packet_size

packet_size (ULONG)

The size, in bytes, of this packet.

TCEREREQUEST Field - packet_revision_number

packet_revision_number (ULONG)

A long integer value that indicates the revision level of the parameter packet.

WPOS_RELEASE_1 2

Indicates OS/2 Warp (PowerPC Edition) Release 1-level parameter packet.

TCEREREQUEST Field - major_event_code

major_event_code (ULONG)

The major event code of the event to be logged (must be between 3 and 4096).

TCEREREQUEST Field - minor_event_code

minor_event_code (ULONG)

The minor event code of the event to be logged (must be between 1 and 65535).

TCEREREQUEST Field - event_data_length

event_data_length (ULONG)

The length of the data, in bytes, that is contained within the caller's event data buffer.

Event data will be truncated to 2044 bytes in length. To trace event data larger than this, break up the data and issue additional [TraceCreateEntry](#) calls.

TCEREREQUEST Field - event_data

event_data (PVOID)

The pointer to a buffer that contains the event data to be logged. The format of the data is specific to each tracepoint.

Event Tracing Sample

```
#define INCL_BASE
#define INCL_DOSMISC
#define INCL_LOGGING
#define INCL_UNI
#define INCL_OS2SERVER_UNI
#ifdef INCL_ULS_UNI
#undef INCL_ULS_UNI
#endif

#include <stddef.h> /* C library of standard defines */
#include <stdlib.h> /* C library of standard routines */
#include <string.h> /* C library for string operations */
#include <stdio.h> /* C library for standard I/O */
#include <ctype.h> /* C library for character operations */
#include <os2.h> /* for all dos api calls */
#include <os2u.h> /* for all dos unicode api calls */
#include <basemid.h> /* Base msgid definitions */
#include <unidef.h> /* Unicode support */
#include <lfdef.h> /* Logging Framework support */

#define QUERY_SIZE 1000
#define BUFFER_SIZE 8192

BOOL SetupDone = FALSE; /* Unicode setup performed flag */

ULONG CpList[8]; /* List of code page info. */
ULONG CpSize; /* Length of code page list. */

UniChar code_page[10]; /* Code page for current OS/2 process */
UniChar locale[128]; /* Locale name */

UconvObject convert_object; /* Unicode conversion object */
LocaleObject locale_object; /* Locale object */

/*-----*/
/* FUNCTION PROTOTYPES */
/*-----*/
/* Unicode support */
/*-----*/
APIRET ConvertToUni(UniChar *UniString, char *CharString, int UniLength);
APIRET ConvertFromUni(UniChar *UniString, char *CharString, int CharLength);
APIRET SetupUni(void);

int main(int argc, char *argv[])
{
    BYTE FmtBuffer[BUFFER_SIZE],
        ReadBuffer[BUFFER_SIZE];

    char *pInsert;

    char DetailStr[256];
    char CvtDetailStr[256];

    ULONG i = 0,
        ulDetailCount = 0,
        ulUniFilesize = 0,
        ulDetailLen = 0,
        ulReadBufferLen = BUFFER_SIZE,
        ulFormatBufferSize = BUFFER_SIZE;

    APIRET rc = NO_ERROR;

    UniChar UniFilename[128];

    LOFREQUESTTRACE OpenPkt;
    LCFREQUESTTRACE ClosePkt;
    LFEREQUESTTRACE FormatPkt;
```



```

LRREQUESTTRACE    ReadPkt;

PLFDETAIL2        pDet2;
PFILTERBLOCK      pFilterBlock;    /* Pointer to start of filters    */

if (argc == 1) {
    printf("Trace inferred filename required\n");
    return(1);
}

/*****
/* Open the file
*****/
ConvertToUni(UniFilename, argv[1], 128);
ulUniFilesize = UniStrlen(UniFilename);

OpenPkt.packet_size      = sizeof(LOFREQUESTTRACE);
OpenPkt.packet_revision_number = WPOS_RELEASE_1;
OpenPkt.filename_length  = &ulUniFilesize;
OpenPkt.filename         = UniFilename;
OpenPkt.FileHandle       = NULLHANDLE;

rc = LogOpenFile(EVENT_TRACE_SERVICE, &OpenPkt);
if (rc != NO_ERROR) {
    printf("LogOpenFile RC(%d)\n", rc);
    exit(rc);
}

ClosePkt.packet_size      = sizeof(LCFREQUESTTRACE);
ClosePkt.packet_revision_number = WPOS_RELEASE_1;
ClosePkt.FileHandle       = OpenPkt.FileHandle;

FormatPkt.number_of_detail_records = &ulDetailCount;
FormatPkt.packet_size              = sizeof(FormatPkt);
FormatPkt.packet_revision_number    = WPOS_RELEASE_1;
FormatPkt.log_entry_buffer          = &ReadBuffer;
FormatPkt.string_buffer_length      = &ulFormatBufferSize;
FormatPkt.string_buffer             = (UniChar *) &FmtBuffer;
FormatPkt.pathname_table_size       = 0;
FormatPkt.pathname_table            = NULL;
FormatPkt.flags                    = TRACE_ENTRY_DATA;

pFilterBlock = NULL;

ReadPkt.packet_size      = sizeof(LRREQUESTTRACE);
ReadPkt.packet_revision_number = WPOS_RELEASE_1;
ReadPkt.flags            = TRACE_READ_SINGLE_RECORD;
ReadPkt.pFilter          = pFilterBlock;
ReadPkt.pLogEntryBuffer  = (UniChar *) ReadBuffer;
ReadPkt.pLogEntryBufferLength = &ulReadBufferLen;
ReadPkt.FileHandle       = OpenPkt.FileHandle;
ReadPkt.entry_ID         = TRACE_NEWEST_ENTRY;

do {
    /*****
    /* Read in one buffer.
    *****/
    ulReadBufferLen = BUFFER_SIZE;
    rc = LogReadEntry(EVENT_TRACE_SERVICE, &ReadPkt);
    if ((rc != NO_ERROR) && (rc != INVALID_LF_AT_END_OF_LOG)) {
        printf("LogReadEntry RC(%d)\n", rc);
    }

    /*****
    /* Loop through the buffer to format one entry at a time
    *****/
    else if ((rc != INVALID_LF_AT_END_OF_LOG) && (ulReadBufferLen > 0)) {

        /*****
        /* Format the data
        *****/
        ulFormatBufferSize = BUFFER_SIZE;
        FormatPkt.log_entry_buffer = (PULONG) &ReadBuffer;
        rc = LogFormatEntry(EVENT_TRACE_SERVICE, &FormatPkt);
        if (rc != NO_ERROR) {
            printf("LogFormatEntry RC(%d)\n", rc);
        }

        /*****
        /* Close the log file
        *****/
        rc = LogCloseFile(EVENT_TRACE_SERVICE, &ClosePkt);
        if (rc != NO_ERROR) {

```

```

        printf("LogCloseFile RC(%d)\n", rc);
    }
    exit(rc);
}

pDet2 = (PLFEDETAIL2) FmtBuffer;
pInsert = (char *) pDet2 + sizeof(LFEDETAIL2);

for (i = 0; i < ulDetailCount; ++i) {
    memset(DetailStr, 0, sizeof(DetailStr));
    ulDetailLen = pDet2->length-sizeof(LFEDETAIL2);
    memcpy(DetailStr, pInsert, ulDetailLen);

    switch (pDet2->type) {
        case TRACE_ENTRY_TIME_TYPE:
            printf("Timestamp                = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_ID_TYPE:
            printf("Entry number                = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_PERSONALITY_TYPE:
            printf("Personality type                = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_MK_TASK_TYPE:
            printf("Microkernel task type          = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_MK_THREAD_TYPE:
            printf("Microkernel thread type        = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_OS2_PROCESS_TYPE:
            printf("OS/2 process type              = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_OS2_THREAD_TYPE:
            printf("OS/2 thread type               = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_MODULE_PATHNAME_TYPE:
            printf("Module pathname                = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_DATA_TYPE_TYPE:
            printf("Entry data type                = %s\n", DetailStr);
            break;

        case BUFFER1_TYPE:
            if (ulDetailLen > 0)
                printf("Entry data                      = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_MAJOR_CODE_TYPE:
            printf("Major code                     = %s\n", DetailStr);
            break;

        case TRACE_ENTRY_MINOR_CODE_TYPE:
            printf("Minor code                     = %s\n", DetailStr);
            break;

        default:
            printf(">> DEFAULT                = %s\n", DetailStr);
            break;
    } /* switch */

    pDet2 = (PLFEDETAIL2) ((char *)pDet2 + pDet2->length);
    pInsert = (char *)pDet2 + sizeof(LFEDETAIL2);
} /* Detail loop */

printf("\n");

} /* Buffer has data */
} while ((rc == NO_ERROR));

/*****
/* Close the log file */
*****/
rc = LogCloseFile(EVENT_TRACE_SERVICE, &ClosePkt);
if (rc != NO_ERROR) {
    printf("LogCloseFile RC(%d)\n", rc);
}

```

```

    exit(rc);
}

UniFreeLocaleObject(locale_object);
UniFreeUconvObject(convert_object);
exit(0);
} /* main */

/***** START OF SPECIFICATION *****/
/* Name:      ConvertToUni */
/* Purpose:    Convert a char string to unicode string. */
/* Returns:    RC from API's */
/* Parameters:
/*     UniChar*   Pointer to converted unicode string
/*     char*      Pointer to string to convert
/*     int        length of Unicode buffer in UniChars
*****/
/***** END OF SPECIFICATION *****/
APIRET ConvertToUni(UniChar *UniString, char *CharString, int UniLength)
{
    APIRET    rc = NO_ERROR;
    char*     pinbuf;
    UniChar*  puni;
    size_t    inbytes, unichars, noni;

    if (SetupDone == FALSE) {
        rc = SetupUni(); /* Setup local and convert objects */
    }

    if (rc == NO_ERROR) {

        /*****
        /* UniUconvToUcs will change the pointers given to it, so send in
        /* temporary pointers
        *****/
        inbytes = strlen(CharString) + 1;
        unichars = UniLength;
        pinbuf = CharString;
        puni = UniString;
        memset(puni, 0, unichars * sizeof(UniChar));

        rc = UniUconvToUcs(convert_object,
                           (void *)&pinbuf,
                           &inbytes,
                           &puni,
                           &unichars,
                           &noni);
    }
    return(rc);
} /* ConvertToUni */

/***** START OF SPECIFICATION *****/
/* Name:      ConvertFromUni */
/* Purpose:    Convert a unicode string to char string. */
/* Returns:    0 successfull conversion
/*             1 conversion failed
/* Parameters:
/*     UniChar*   UniString; Pointer to string to convert
/*     char*      CharString; Pointer to converted string
/*     int        CharLength; length of the char buffer
*****/
/***** END OF SPECIFICATION *****/
APIRET ConvertFromUni(UniChar *UniString, char *CharString, int CharLength)
{
    APIRET    rc = NO_ERROR;
    char*     poutbuf;
    UniChar*  puni;
    size_t    unichars, chars, noni;

    if (SetupDone == FALSE) {
        rc = SetupUni(); /* Setup local and convert objects */
    }

    if (rc == NO_ERROR) {

        /*****
        /* UniUconvFromUcs will change the pointers given to it, so send in
        *****/

```

```

/* temporary pointers */
/*****
unichars = UniStrlen(UniString);
chars    = CharLength;
poutbuf  = CharString;
puni     = UniString;
memset(poutbuf, 0, chars);          /* Zero output buffer and null term */

rc = UniUconvFromUcs(convert_object, /* Conversion Object */
                    &puni,          /* Uni Input Buffer addr */
                    &unichars,      /* Bytes left in ucsbuf */
                    (void *)&poutbuf, /* Output Buffer address */
                    &chars,         /* Bytes left in outbuf */
                    &noni);         /* nonident conversions */
}
return(rc);
} /* ConvertFromUni */

/***** START OF SPECIFICATION *****/
/* Name:      SetupUni */
/* Purpose:   Setup local and convert objects for unicode conversions */
/* Returns:   RC from API's used */
/* Parameters: none */
/***** END OF SPECIFICATION *****/
APIRET SetupUni(void)
{
    APIRET rc = NO_ERROR;

    rc = UniCreateLocaleObject(UNI_UCS_STRING_POINTER, L"", &locale_object);

    /*****
    /* Get the value of LC_MESSAGES to determine which language */
    /*****
    rc = UniQueryLocaleObject(locale_object,
                            LC_MESSAGES,
                            UNI_UCS_STRING_POINTER,
                            (PVOID)&locale);

    if (rc != NO_ERROR)
        return(rc);

    UniStrcpy(code_page, L"IBM-850");

    rc = UniCreateUconvObject(code_page, &convert_object);
    if (rc != NO_ERROR)
        return(rc);

    SetupDone = TRUE;
    return(rc);
} /* SetupUni */

```

Performance Services

Defining a metric as individual units of information, a provider is any subsystem that contains metrics for export, a consumer is any subsystem that needs to import metrics, and a method is the process by which metric information is retrieved. The purpose of Performance Services is to isolate the consumer from method changes within the PNS, microkernel and hardware, so the consumer can continue to use the same architected interface to obtain information.

The level of Performance Services support on the system can be obtained by [dcilInitialize](#).

This chapter includes the following sections:

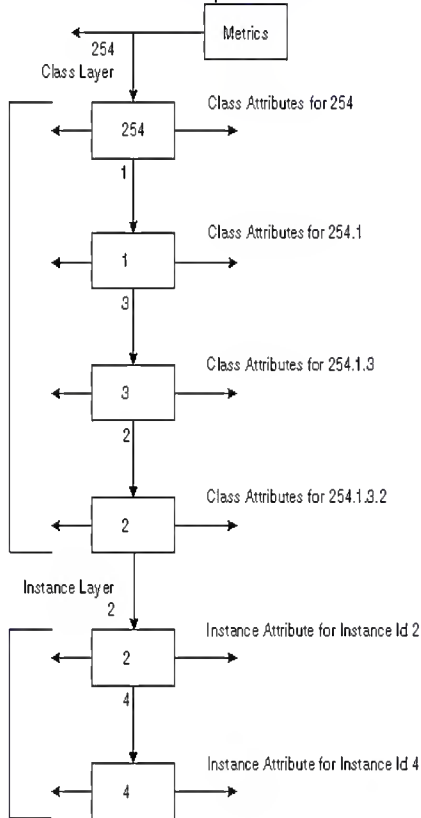
- [Metrics Name Space](#)
- [DCI Functions](#)
- [Return Status and Structures](#)
- [DCI Common Functions](#)
- [DCI Consumer Functions](#)
- [DCI Provider Functions](#)
- [DCI Data Types](#)

Metrics Name Space

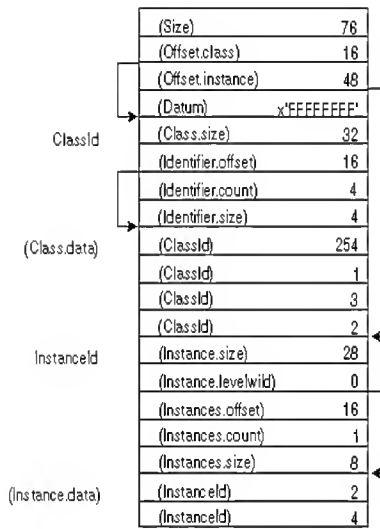
The Data Capture Interface (DCI) supports a hierarchical name space that is used to uniquely identify specific instances of available metrics. All metrics are registered and their details and methods are kept in the name space. Metric consumers use the metrics name or identifier in the name space when reading or requesting metric data.

The metrics name space is grouped into two layers with as many levels per layer as necessary: one layer to identify a metric class and a lower layer to identify the available instances of that metric. The first layer is called a metric class identifier and the second an instance identifier. Together the two name space layers uniquely identify a metric instance. The data types for these three definitions are **DCIClassId** for the class identifier, **DCIInstanceId** for the instance identifier, and **DCIMetricId** for the combination of the two commonly called metric identifier.

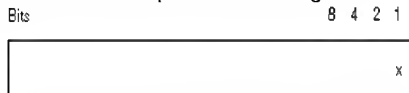
The following figure shows the name space representation for an example Class Id 254.1.3.2 with an Instance Id Level of 2.4.



The following figure shows an example of a filled in MetricId with a two-level InstanceId.



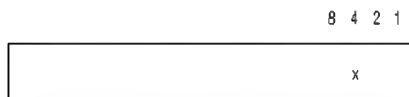
Multiple instances can be represented within a single MetricId. Wildcarding is done via a bit mask in the InstanceId structure. For example, InstanceId level one uses the mask 1, InstanceId level two uses the mask 2, and InstanceId level three uses the mask 4. Masks for multiple levels can be OR'd together. The following figure shows an example of wildcarding.



Bit position 0 corresponds to a decimal value of 1 which means the first instance level is being wildcarded and all instances at this level are being chosen.



Bit position 1 corresponds to a decimal value of 2 which means the second instance level is being wildcarded and all instances at this level are being chosen.



Bit position 2 corresponds to a decimal value of 4 which means the third instance level is being wildcarded and all instances at this level are being chosen.

Note: For Release 1, wildcarding is only supported for the last level of a class or an instance level.

DCI Functions

The general functions from a consumer point of view are:

dcInitialize

Returns the versionLevel of the Performance Services, establishes a Data Capture Interface (DCI) connection.

dcOpen

Opens a session and determines what classes of metrics are to be returned via later dcGetData calls. A 4-byte session key is returned. For each class it marks what session key opened it, if any. dcOpen loads the appropriate libraries and/or Performance Server for the metrics chosen if they are not already loaded.

dciGetData	Returns a buffer of data for a given session.
dciClose	Clears the session entries for the inputted key. Unloads the appropriate provider-shared libraries and/or Performance Server if this is the last session using the metric.
dciTerminate	Terminates the Data Capture Interface connection.
dciFree	Returns storage obtained by the Performance Services.
dciListClassId	Obtains metric class identifiers registered in the metrics name space.
dciListInstanceld	Obtains a list of instance identifiers registered in the metrics name space. Loads the appropriate libraries and/or Performance Server for the metrics chosen.
dciGetClassAttributes	Obtains class attribute information from the name space.
dciGetInstAttributes	Obtains metric instance attributes from the name space. Loads the appropriate libraries and/or Performance Server for the metrics chosen.

All entry and exits from the Consumers APIs have trace entry and exit points using the major code of 158 and the minor code listed in the PS_TRCHKID.H file.

The general functions from a provider point of view are:

dciInitialize	Returns the VersionLevel of the Performance Services, establishes a DCI connection and registers the metrics of the microkernel in the name space.
dciTerminate	Terminates the DCI connection.
dciRegister	Registers a list of metrics.
dciUnregister	Unregisters a list of metrics.
dciAddInstance	Adds an instance or a list of instances to a metric.
dciRemoveInstance	Removes an instance or a list of instances from a metric.
architected SPIs	Receive requests from the Performance Services for information needed by consumers.

Provider metrics are registered in the name space.

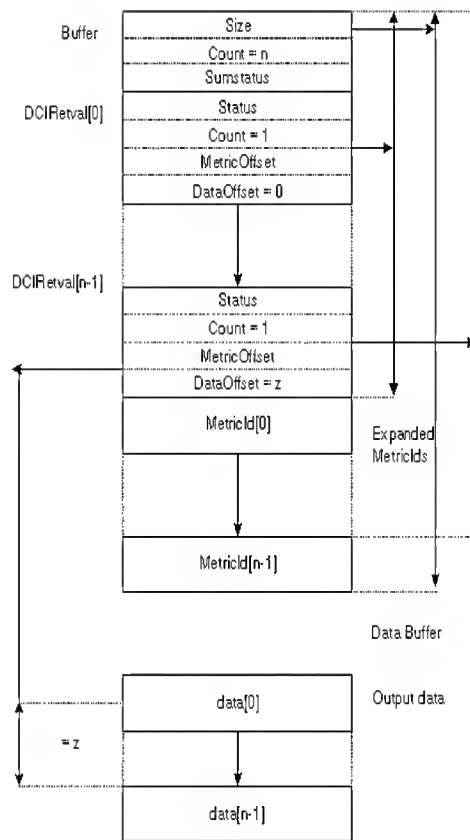
All entry and exits from the Provider APIs have trace entry and exit points using the major code of 158 and the minor code listed in the PS_TRCHKID.H file.

Return Status and Structures

All DCI functions except [dciEZRegister](#) return [DCIStatus](#). DCIStatus returns success when a [DCIRetval](#) structure has been created. The summary status of [DCIReturn](#) and the [DCIRetval](#) structures must be checked for overall status and individual metric status. The error status value is a summary error and may not be the same for every metric in the input list. Having an error status does not imply nothing was returned. The application must traverse the [DCIReturn](#) structure to discover the status value for each metric in the input list.

The data portion of the return buffer can be a separate buffer whose storage is acquired by the DCI via [vm_allocate](#) and must be returned by the caller via [dciFree](#), just as the return structure is. Where a [dataBuffer](#) is used, the [size](#) field in the [DCIReturn](#) structure in the status buffer does not include the size of the data area. A separate data buffer is used because volatile instances make the size of the return structure unknown until all of the data is returned for all of the instances.

The following figure shows an example of separate buffers used to return [DCIRetval](#), [MetricId](#), and the output data.



The return structure begins with a four byte integer that contains the total number of bytes written. The purpose of this size is to allow range checking and for quick traversal of multiple [DCIReturn](#) structures. The total size is followed by a count of the total number of output list identifiers. This count could differ from the input list of identifiers due to a wildcard expansion.

This header, the total size and count plus the summary status is followed by two arrays. The first is an array of count [DCIRetval](#) structures, one for each expanded metric or class identifier. The [DCIRetval](#) structure gives the return status, a count of the number of returned data items for the request (which is set to 1 due to one Retval per metricid/classid expansion), an offset to the metric identifier or class identifier which may have been expanded from a wildcard if supported by the command, and an offset to the output data for this request. The metric identifier is an offset from the beginning of the [DCIReturn](#), and the data offset is an offset from the beginning of dataBuffer structure when a separate data buffer is used (otherwise it's an offset from the beginning of [DCIReturn](#)). Note that the [DCIRetval](#) structure is a fixed size allowing for quick traversal of the DCIRetval array.

Following the array of [DCIRetval](#) structures is the metric identifier or class identifier which may have been expanded from a wildcard if supported by the command. Some DCI routines, take class identifier lists as input while others take full metric identifiers as input. Applications acquire offsets into these structures by traversing the DCIRetval array.

Some routines may not return data. In this case there is no returned data buffer and the area in the [DCIReturn](#) structure and the *count* and *dataOffset* fields of the [DCIRetval](#) structure are unused and their values are set to zero. Clearing these fields prevents applications which inadvertently ignore status values from access memory outside of the input buffer range.

DCI Common Functions

This sections describes the following Data Capture Interface (DCI) functions that are used by both Consumer and Provider applications:

- [dcIInitialize](#)
- [dcITerminate](#)

dcIInitialize

dcilInitialize - Syntax

Initializes a connection to the Data Capture Interface.

```
#include <dc_i.h>

char      **ps_version;
DCIStatus  ulrc;      /* Return code. */

ulrc = dcilInitialize(ps_version);
```

dcilInitialize Parameter - ps_version

ps_version (char **) - in/out
Value of the version of the DCI on the system.

dcilInitialize Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dcilInitialize returns one of the following values:

DCI_SUCCESS

A DCI connection was established to a metric consumer or provider.

DCI_SYSEERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

dcilInitialize - Parameters

ps_version (char **) - in/out
Value of the version of the DCI on the system.

ulrc ([DCIStatus](#)) - returns
Return code.

dcilInitialize returns one of the following values:

DCI_SUCCESS

A DCI connection was established to a metric consumer or provider.

DCI_SYSEERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of

the application.

dcilInitialize - Remarks

This interface is used by metric consumers and providers.

dcilInitialize is used to perform implementation-specific initialization of a DCI implementation prior to performing DCI transactions. Conversely, [dcilTerminate](#) is used to perform implementation-specific termination. The version level of the code is returned as output from the dcilInitialize call.

dcilInitialize - Related Functions

- [dcilTerminate](#)
-

dcilInitialize - Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

main(int argc, char **argv, char **envp)
{
    char                *ps_version;
    DCIStatus           status;
    DCIMetricId         *metric = NULL;
    DCIClassId          *class = NULL;
    DCIInstanceId        *instance = NULL;
    DCIReturn           *retBuffer = NULL;
    DCIHandle           key;
    uint32              *class_data = NULL;
    uint32              *inst_data = NULL;
    uint32              numIds;
    uint32              RetBufSize = 0;
    uint32              flags = 0;

    console_printf("ENTERING:main\n");

    /* Initialize dci services */
    status = dcilInitialize(&ps_version);
    console_printf("dcilInitialize status=0x%x\n", status);
    if(status != DCI_SUCCESS)
    {
        console_printf("dcilInitialize FAILED\n");
        console_printf("EXITING #0:main\n");
        exit(-1);
    }
    console_printf("VERSION=%s\n", ps_version);

    /* Prepare the metric id structure */
    metric = (DCIMetricId *)malloc(72);
    metric->size = 72;
    metric->classId.offset = 16;
    metric->instanceId.offset = 48;
    metric->datumId = 255;

    class = (DCIClassId *)&(metric->data);
    class->size = 32;
    class->identifier.offset = 16;
```

```

class->identifier.count = 4;
class->identifier.size = 4;
class_data = (uint32 *)&(class->data);
*class_data++ = 254;
*class_data++ = 1;
*class_data++ = 1;
*class_data = 1;

instance = (DCIInstanceId *)((char *)class + class->size);
instance->size = 24;
instance->levelWild = 0;
instance->instances.offset = 20;
instance->instances.size = 4;
instance->instances.count = 1;
inst_data = (uint32 *)&(instance->data);
*inst_data = 0;

/* Call dciOpen */
retBuffer = NULL;
RetBufSize = 0;
numIds = 1;
status = dciOpen(&key, metric, numIds, &retBuffer, RetBufSize, flags);
free(metric);
if(status != DCI_SUCCESS) /* if FAILURE */
{
    console_printf("dciOpen FAILED. status 0x%x\n",status);
    status = dciTerminate();
    console_printf("dciTerminate status = 0x%x\n",status);
    console_printf("EXITING #1 main\n");
    exit(-1);
}
console_printf("dciOpen PASSED. status 0x%x - summmmary status 0x%x\n",
               status, retBuffer->sumstatus);
if( retBuffer->sumstatus != DCI_SUCCESS )
    console_printf("There was atleast one FAILURE in the return
buffer\n");

/* Close the key, terminate the DCI services and Clean up */
status = dciFree((uint32)retBuffer, retBuffer->size);
console_printf("dciFree status 0x%x\n",status);
status = dciClose(key);
console_printf("dciClose status = 0x%x\n",status);
status = dciTerminate();
console_printf("dciTerminate status = 0x%x\n",status);
console_printf("EXITING #2:main\n");
exit(0);
}

```

dcilInitialize - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

dciTerninate

dciTerninate - Syntax

Terminates a connection to the Data Capture Interface.

```
#include <dc_i.h>

DCIStatus    ulrc; /* Return code. */

ulrc = dc_iTerminate();
```

dc_iTerminate Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dc_iTerminate returns one of the following values:

DCI_SUCCESS	The DCI connection was successfully terminated.
DCI_NOTINITIALIZED	The requestor has not yet initialized the DCI connection.

dc_iTerminate - Parameters

ulrc ([DCIStatus](#)) - returns
Return code.

dc_iTerminate returns one of the following values:

DCI_SUCCESS	The DCI connection was successfully terminated.
DCI_NOTINITIALIZED	The requestor has not yet initialized the DCI connection.

dc_iTerminate - Remarks

This interface is used by metric providers and consumers. dc_iTerminate allows DCI implementations to know when an application is ending a series of DCI transactions. This interface allows the DCI implementation to perform implementation specific termination. Conversely, [dc_iInitialize](#) is used to perform implementation specific initializations. This function resets the dc_iInitialize variable in the task's address space.

No output buffers are returned from the execution of this command.

dc_iTerminate - Related Functions

- [dc_iInitialize](#)

dciTerninate - Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

main(int argc, char **argv, char **envp)
{
    char                *ps_version;
    DCIStatus           status;
    DCIMetricId         *metric = NULL;
    DCIClassId          *class = NULL;
    DCIInstanceId       *instance = NULL;
    DCIReturn           *retBuffer = NULL;
    DCIHandle           key;
    uint32              *class_data = NULL;
    uint32              *inst_data = NULL;
    uint32              numIds;
    uint32              RetBufSize = 0;
    uint32              flags = 0;

    console_printf("ENTERING:main\n");

    /* Initialize dci services */
    status = dciInitialize(&ps_version);
    console_printf("dciInitialize status=0x%x\n", status);
    if(status != DCI_SUCCESS)
    {
        console_printf("dciInitialize FAILED\n");
        console_printf("EXITING #0:main\n");
        exit(-1);
    }
    console_printf("VERSION=%s\n", ps_version);

    /* Prepare the metric id structure */
    metric = (DCIMetricId *)malloc(72);
    metric->size = 72;
    metric->classId.offset = 16;
    metric->instanceId.offset = 48;
    metric->datumId = 255;

    class = (DCIClassId *)&(metric->data);
    class->size = 32;
    class->identifier.offset = 16;
    class->identifier.count = 4;
    class->identifier.size = 4;
    class_data = (uint32 *)&(class->data);
    *class_data++ = 254;
    *class_data++ = 1;
    *class_data++ = 1;
    *class_data = 1;

    instance = (DCIInstanceId *)((char *)class + class->size);
    instance->size = 24;
    instance->levelWild = 0;
    instance->instances.offset = 20;
    instance->instances.size = 4;
    instance->instances.count = 1;
    inst_data = (uint32 *)&(instance->data);
    *inst_data = 0;

    /* Call dciOpen */
    retBuffer = NULL;
    RetBufSize = 0;
    numIds = 1;
    status = dciOpen(&key, metric, numIds, &retBuffer, RetBufSize, flags);
    free(metric);
    if(status != DCI_SUCCESS) /* if FAILURE */
    {
        console_printf("dciOpen FAILED. status 0x%x\n", status);
        status = dciTerminate();
        console_printf("dciTerminate status = 0x%x\n", status);
        console_printf("EXITING #1 main\n");
        exit(-1);
    }
}
```

```

    }
    console_printf("dciOpen PASSED. status 0x%x - summary status 0x%x\n",
                  status, retBuffer->sumstatus);
    if( retBuffer->sumstatus != DCI_SUCCESS )
        console_printf("There was atleast one FAILURE in the return
buffer\n");

    /* Close the key, terminate the DCI services and Clean up */
    status = dciFree((uint32)retBuffer, retBuffer->size);
    console_printf("dciFree status 0x%x\n",status);
    status = dciClose(key);
    console_printf("dciClose status = 0x%x\n",status);
    status = dciTerminate();
    console_printf("dciTerminate status = 0x%x\n",status);
    console_printf("EXITING #2:main\n");
    exit(0);
}

```

dciTerninate - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DCI Consumer Functions

This sections describes the following Data Capture Interface (DCI) functions used by Consumer applications:

- [dciClose](#)
- [dciFree](#)
- [dciGetClassAttributes](#)
- [dciGetData](#)
- [dciGetInstAttributes](#)
- [dciListClassId](#)
- [dciListInstanceId](#)
- [dciOpen](#)

dciclose

dciclose - Syntax

Closes a metrics list

```
#include <dci.h>
```

```
DCIHandle    handle;  
DCIStatus    ulrc;    /* Return code. */  
  
ulrc = dciClose(handle);
```

dciclose Parameter - handle

handle (DCIHandle) - input
Handle that was returned from a prior dciOpen call.

dciclose Return Value - ulrc

ulrc (DCIStatus) - returns
Return code.

dciclose returns one of the following values:

DCI_SUCCESS	The operation was a success.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_SYSERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_BADHANDLE	The handle provided in not currently open.

dciclose - Parameters

handle (DCIHandle) - input
Handle that was returned from a prior dciOpen call.

ulrc (DCIStatus) - returns
Return code.

dciclose returns one of the following values:

DCI_SUCCESS	The operation was a success.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_SYSERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_BADHANDLE	The handle provided in not currently open.

dciclose - Remarks

dciclose is the counterpart of [dciOpen](#). dciClose cleans up the session information kept by the Performance Services, unloads the appropriate

libraries and/or Performance Server, if appropriate, and closes the handle that is associated with a list of metrics. Subsequent attempts to use the closed handle will fail. The Performance Services closes the notify with the name space for any of the non-volatile metrics opened as part of this session. No `bufferAddress` or `dataAddress` is used to return any information back to the consumer.

dciclose - Related Functions

- [dciclose](#)
-

dciclose - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

dcifree

dcifree - Syntax

Returns storage allocated by the Performance Server for buffers used for dci APIs.

```
#include <dci.h>

uint32      bufferAddress;
uint32      bufferSize;
DCIStatus   ulrc;          /* Return code. */

ulrc = dcifree(bufferAddress, bufferSize);
```

dcifree Parameter - bufferAddress

bufferAddress (uint32) - input
Address of a returned buffer previously allocated by the Performance Server.

dcifree Parameter - bufferSize

bufferSize (uint32) - input
The size of the returned buffer.

dcifree Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dcifree returns one of the following values:

DCI_SUCCESS	The operation was a success.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize.

dcifree - Parameters

bufferAddress (uint32) - input
Address of a returned buffer previously allocated by the Performance Server.

bufferSize (uint32) - input
The size of the returned buffer.

ulrc ([DCIStatus](#)) - returns
Return code.

dcifree returns one of the following values:

DCI_SUCCESS	The operation was a success.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize.

dcifree - Remarks

dcifree returns storage obtained by the Performance Server for the client using the dci APIs. This routine uses as input a buffer address, **bufferAddress* , and a buffer size, *bufferSize* , to deallocate the storage using the proper machine dependent routines. This routine does not return a buffer address or a data address.

dcifree - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

dcigetClassAttributes

dcigetClassAttributes - Syntax

Acquires metric class attributes.

```
#include <dcI.h>

DCIHandle      handle;
DCIClassId     *classIdList;
uint32         numIds;
DCIReturn      **bufferAddress;
uint32         bufferSize;
DCIStatus      ulrc;          /* Return code. */

ulrc = dcigetClassAttributes(handle, classIdList,
                             numIds, bufferAddress, bufferSize);
```

dcigetClassAttributes Parameter - handle

handle (DCIHandle) - input

Handle that was returned from a prior [dcIOpen](#) call for a prior open of *classIdList* or a superset of *classIdList* . This field is optional. If set to 0 the argument will be ignored.

dcigetClassAttributes Parameter - classIdList

classIdList (DCIClassId *) - input

Address of a list of metric class identifiers. This field is optional. If set to 0 the argument will be ignored.

dcigetClassAttributes Parameter - numIds

numIds (uint32) - input

The number of input metric class identifiers.

dcigetClassAttributes Parameter - bufferSize

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using [dcifree](#).

dcigetClassAttributes Parameter - bufferSize

bufferSize (uint32) - input
Reserved, set to zero.

dcigetClassAttributes Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dcigetClassAttributes returns one of the following values:

- DCI_SUCCESS
The [DCIReturn](#) structure has been written to the output buffer.
 - DCI_NOTINITIALIZED
The requester has not yet initialized the DCI.
 - DCI_SYSEERROR
A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
 - DCI_INVALIDARG
One of the input arguments is invalid. For example, a negative value was used for *numIds* or *bufferSize* . Also, *metricIdList* could not be read or the buffer could not be written.
 - DCI_BADHANDLE
The handle provided is not currently open.
-

dcigetClassAttributes - Parameters

handle ([DCIHandle](#)) - input
Handle that was returned from a prior [dcioopen](#) call for a prior open of *classIdList* or a superset of *classIdList* . This field is optional. If set to 0 the argument will be ignored.

classIdList ([DCIClassId *](#)) - input
Address of a list of metric class identifiers. This field is optional. If set to 0 the argument will be ignored.

numIds (uint32) - input
The number of input metric class identifiers.

bufferAddress ([DCIReturn **](#)) - output

Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using [dciFree](#).

bufferSize (uint32) - input
Reserved, set to zero.

ulrc ([DCIStatus](#)) - returns
Return code.

dciGetClassAttributes returns one of the following values:

DCI_SUCCESS	The DCIReturn structure has been written to the output buffer.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for <i>numIds</i> or <i>bufferSize</i> . Also, <i>metricIdList</i> could not be read or the buffer could not be written.
DCI_BADHANDLE	The handle provided is not currently open.

dciGetClassAttributes - Remarks

dciGetClassAttributes is used to retrieve the attribute structures for a list of class identifiers. The class identifiers can be optionally wildcarded.

An optional handle may be provided for the *classIdList* . If the handle is valid and if *classIdList* . contains wildcarded classes, then all metric classes of *classIdList* . are confirmed to be a subset of the opened metric classes represented by the handle, and any changed classes will be marked with a special status, DCI_CLASSES_CHANGED, in the associated [DCIRetval](#) structure. The *classIdList* . is a subset of the opened metrics associated with handle. If the handle provided is 0 then no such confirmation or status is provided. Not specifying a handle is useful if an application traversing the namespace wants to examine attributes, such as the label, without calling [dciOpen](#). A handle should be specified if attributes are being retrieved prior to data collection because it further validates the metric list prior to obtaining data for it. The *classIdList* . can be a subset of the metrics opened with the handle.

dciGetClassAttributes determines if each classid in the *classIdList* . is currently registered in the metrics name space, expands any class wildcards in the classOffset buffer, and determines if each expanded metric class is currently registered in the metrics name space. For each expanded metric class a [DCIRetval](#) references the expanded [DCIClassId](#) structure and if available, the *dataOffset* member of [DCIRetval](#) references a [DCIClassAttr](#) for the associated, registered metric class. This routine can be used with class level wildcarding. Any changes to the registration since the handle was created for the classIdList (volatile instances excluded) will be indicated by an appropriate status return value.

The summary status of all individual [DCIRetval](#) structure status members is stored in the [DCIReturn](#) structure status member. This summary status represents the highest severity of status returned among all [DCIRetval](#) structures.

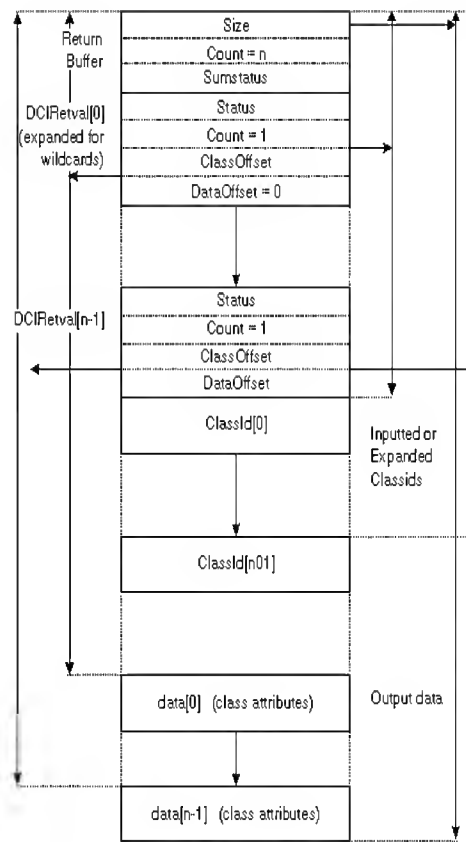
The summary status may be one of the following values:

DCI_SUCCESS	All status returned was successful.
DCI_FAILURE	There was at least one failure status.

For each [DCIRetval](#) structure returned, the status member may contain one of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
DCI_NOCLASS	The Class is not present in the name space.
DCI_CLASSECHANGED	This new class has been added within the scope of a wildcarded class list.

The following figure shows the return structures for dciGetClassAttributes.



dcigetClassAttributes - Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

main(int argc, char **argv, char **envp)
{
    char          *ps_version;
    DCIStatus     status;
    DCIHandle     key = 0;
    DCIReturn     *retBuffer = NULL;
    DCIClassId    *class = NULL;
    uint32        numIds = 1;
    uint32        RetBufSize = 0;
    uint32        *class_data;

    console_printf("ENTERING:main\n");

    /* Initialize dci services */
    status = dciInitialize(&ps_version);
    console_printf("dciInitialize status=0x%x\n", status);
    if(status != DCI_SUCCESS)
    {
        console_printf("dciInitialize FAILED\n");
        console_printf("EXITING #0:main\n");
        exit(-1);
    }
    console_printf("VERSION=%s\n", ps_version);

    class = (DCIClassId *)malloc(32);
    class->size = 32;
    class->identifier.offset = 16;
    class->identifier.count = 3;
    class->identifier.size = 4;
```

```

class_data = (uint32 *)&(class->data);
*class_data++ = 254;
*class_data++ = 1;
*class_data++ = -1;
*class_data = 0;

key = 0;
numIds = 1;
status = dciGetClassAttributes(key, class, numIds, &retBuffer, RetBufSize);
free(class);
if(status != DCI_SUCCESS) /* if FAILURE */
{
    console_printf("dciGetClassAttributes FAILED. status 0x%x\n",status);
    status = dciTerminate();
    console_printf("dciTerminate status = 0x%x\n",status);
    console_printf("EXITING #1 main\n");
    exit(-1);
}
console_printf("dciGetClassAttributes PASSED. status 0x%x - summmmary status 0x%x\n", status, retBuffer->sumstatus);
if( retBuffer->sumstatus != DCI_SUCCESS )
    console_printf("There was atleast one FAILURE in the return buffer\n");

/* Terminate the DCI services and Clean up */
status = dciFree((uint32)retBuffer, retBuffer->size);
console_printf("dciFree status 0x%x\n",status);
status = dciTerminate();
console_printf("dciTerminate status = 0x%x\n",status);
console_printf("EXITING #2:main\n");
exit(0);
}

```

dcigetClassAttributes - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

dcigetData

dcigetData - Syntax

Gets metric data.

```

#include <dci.h>

DCIHandle      handle;
DCIMetricID    *metricIdList;
uint32         numIds;
DCIReturn      **bufferAddress;
uint32         bufferSize;
void           **dataAddress;
uint32         *dataSize;
uint64         *timeout;
DCIStatus      ulrc;          /* Return code. */

```

```
ulrc = dciGetData(handle, metricIdList, numIds,  
    bufferAddress, bufferSize, dataAddress,  
    dataSize, timeout);
```

dciGetData Parameter - handle

handle (DCIHandle) - input

Handle that was returned from a prior [dciOpen](#) call for prior open of *metricIdList* or a superset of *metricIdList*.

dciGetData Parameter - metricIdList

metricIdList (DCIMetricID *) - input

Address of a list of metric identifiers.

dciGetData Parameter - numIds

numIds (uint32) - input

The number of input metric identifiers.

dciGetData Parameter - bufferAddress

bufferAddress (DCIReturn **) - output

Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using [dciFree](#)

dciGetData Parameter - bufferSize

bufferSize (uint32) - output

Reserved, set to zero.

dciGetData Parameter - dataAddress

dataAddress - output
Points to a the address of a return data buffer.

dcigetdata Parameter - dataSize

dataSize (uint32 *) - output
Address of the size of the return data buffer.

dcigetdata Parameter - timeout

timeout (uint64 *) - input
Pointer to a timeval structure that specifies the maximum time to wait for polled data delivery. When timeout is a null pointer, dcigetdata blocks indefinitely. The use of this variable is not supported for Release 1.

dcigetdata Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dcigetdata returns one fo the following values:

- DCI_SUCCESS
The [DCIReturn](#) structure has been written to the output buffer.
 - DCI_NOTINITIALIZED
The requester has not yet initialized the DCI.
 - DCI_SYSERROR
A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
 - DCI_INVALIDARG
One of the input arguments is invalid. For example, a negative value was used for *numIds* or *bufferSize* . Also, either *metricIdList* could not be read or the buffer could not be written to *bufferAddress* .
 - DCI_BADHANDLE
The handle provided is not curretlly open.
-

dcigetdata - Parameters

handle (DCIHandle) - input
Handle that was returned from a prior [dcioOpen](#) call for prior open of *metricIdList* or a superset of *metricIdList* .

metricIdList (DCIMetricID *) - input
Address of a list of metric identifiers.

numIds (uint32) - input
The number of input metric identifiers.

bufferAddress ([DCIReturn **](#)) - output

Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using [dciFree](#)

bufferSize (uint32) - output

Reserved, set to zero.

dataAddress - output

Points to the address of a return data buffer.

dataSize (uint32 *) - output

Address of the size of the return data buffer.

timeout (uint64 *) - input

Pointer to a timeval structure that specifies the maximum time to wait for polled data delivery. When timeout is a null pointer, dciGetData blocks indefinitely. The use of this variable is not supported for Release 1.

ulrc ([DCIStatus](#)) - returns

Return code.

dciGetData returns one of the following values:

DCI_SUCCESS

The [DCIReturn](#) structure has been written to the output buffer.

DCI_NOTINITIALIZED

The requester has not yet initialized the DCI.

DCI_SYSERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for *numIds* or *bufferSize*. Also, either *metricIdList* could not be read or the buffer could not be written to *bufferAddress*.

DCI_BADHANDLE

The handle provided is not currently open.

dciGetData - Remarks

dciGetData is used to acquire polled metrics data. A handle for the *metricIdList* must be provided. The *metricIdList* can be a subset of the metrics opened with the handle.

This routine determines if each *classId* and *instanceId* in the *metricIdList* is currently registered in the metrics name space and if the *metricIdList* is a subset of what was requested via [dciOpen](#) for the given handle. The *datumId* in the [DCIMetricId](#) is ignored. If the *metricIdList* pointer is NULL, then the list of metrics chosen at [dciOpen](#) for the given handle will be used.

dciGetData will return a per-metric status in the return buffer using the standard [DCIReturn](#) structure. dciGetData expands any metric class and metric instance wildcards appearing in the *metricIdList* and determines if each expanded metric class and instance is currently registered in the metrics name space and represents a subset of the metrics represented by the handle.

dciGetData accepts both class and instance level wildcards. If wildcards are provided, then status will indicate whether a new class or instance has been added since the last such DCI routine was issued. The application will reference the appropriate [DCIClassAttr](#) structures to determine the size and type of each piece of data, and in the case of the wildcarded datumID, the offset of each piece of data within the whole class of data returned. Any registration changes to the name space since the handle was created for the *metricIdList* (volatile classes and instances excluded) will be indicated by an appropriate status return value.

The timeout parameter is ignored and not supported for Release 1.

For non volatile classes and instances change notification is supported; however, for volatile classes or instances change notification is not supported.

For each expanded metric identifier a [DCIRetval](#) reply is created in the [DCIReturn](#) structure stored in the return buffer pointed to by *bufferAddress*. The *metricOffset* member of each [DCIRetval](#) specifies a location relative to *bufferAddress* and references the expanded [DCIMetricId](#) structure. The *dataOffset* member of each [DCIRetval](#) specifies a location relative to the *dataAddress* argument.

Note: Even though the routine does not return DCI_SUCCESS there can be valid data within one or more [DCIRetval](#) array elements.

The summary status of all individual [DCIRetval](#) structure status members is stored in the [DCIReturn](#) structure status member. This summary status represents the highest severity of status returned among all [DCIRetval](#) structures.

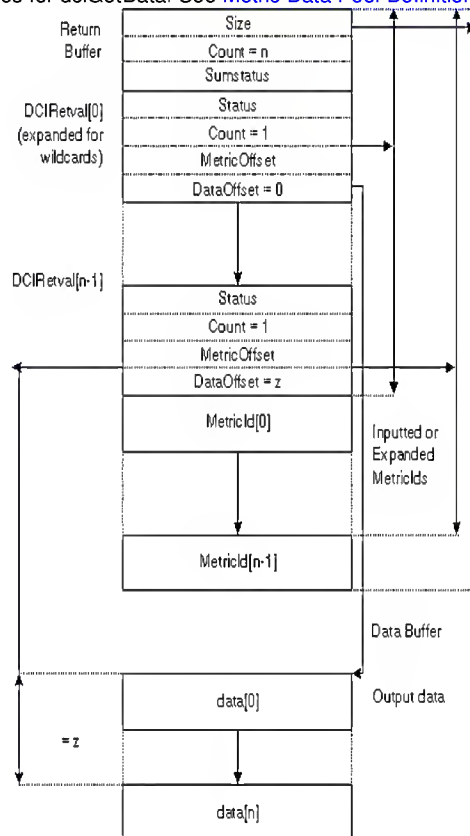
The summary status may be one of the following values:

DCI_SUCCESS	All status returned was successful.
DCI_FAILURE	There was at least one failure status.

For each [DCIRetval](#) structure returned, the status member may contain one of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
DCI_NOCLASS	The class identifier is not present in the name space.
DCI_NOINSTANCE	The requested instance identifier is not in the name space.
DCI_CLASSCHANGED	This new class has been added within the scope of a wildcarded class list.
DCI_INSTANCECHANGED	This new Instance has been added within the scope of a wildcarded class request.

The following figure shows the return structures for dciGetData. See [Metric Data Pool Definition](#) for a definition of the output data.



dciGetData - Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

main(int argc, char **argv, char **envp)
{
    char          *ps_version;
    DCIStatus     status;
```

```

DCIMetricId      *metric = NULL;
DCIClassId       *class = NULL;
DCIInstanceId    *instance = NULL;
DCIReturn        *retBuffer = NULL;
DCIHandle        key;
uint32           *class_data = NULL;
uint32           *inst_data = NULL;
uint32           numIds;
uint32           RetBufSize = 0;
uint32           flags = 0;
uint32           dataSize;
void             *dataAddress = NULL;
UMATimeVal       timx;

console_printf("ENTERING:main\n");

/* Initialize dci services */
status = dciInitialize(&ps_version);
console_printf("dciInitialize status=0x%x\n", status);
if(status != DCI_SUCCESS)
{
    console_printf("dciInitialize FAILED\n");
    console_printf("EXITING #0:main\n");
    exit(-1);
}
console_printf("VERSION=%s\n", ps_version);

/* Prepare the metric id */
metric = (DCIMetricId *)malloc(72);
metric->size = 72;
metric->classId.offset = 16;
metric->instanceId.offset = 48;
metric->datumId = 255;

class = (DCIClassId *)&(metric->data);
class->size = 32;
class->identifier.offset = 16;
class->identifier.count = 4;
class->identifier.size = 4;
class_data = (uint32 *)&(class->data);
*class_data++ = 254;
*class_data++ = 1;
*class_data++ = 1;
*class_data = 1;

instance = (DCIInstanceId *)((char *)class + class->size);
instance->size = 24;
instance->levelWild = 0;
instance->instances.offset = 20;
instance->instances.size = 4;
instance->instances.count = 1;
inst_data = (uint32 *)&(instance->data);
*inst_data = 0;

/* do a dciOpen */
retBuffer = NULL;
RetBufSize = 0;
numIds = 1;
status = dciOpen(&key, metric, numIds, &retBuffer, RetBufSize, flags);
free(metric);
if((status != DCI_SUCCESS) || (retBuffer->sumstatus != DCI_SUCCESS))
{
    if(retBuffer != NULL)
    {
        console_printf("dciOpen FAILED. status 0x%x summary status 0x%x\n",
                        status, retBuffer->sumstatus);
        status = dciFree((uint32)retBuffer, retBuffer->size);
        console_printf("dciFree status 0x%x\n", status);
    }
    else
    {
        console_printf("dciOpen FAILED. status 0x%x\n", status);
        status = dciTerminate();
        console_printf("dciTerminate status = 0x%x\n", status);
        console_printf("EXITING #1 main\n");
        exit(-1);
    }
}
console_printf("dciOpen PASSED. status 0x%x - summmmary status 0x%x\n",
                status, retBuffer->sumstatus);
status = dciFree((uint32)retBuffer, retBuffer->size);
console_printf("dciFree status 0x%x\n", status);

retBuffer = NULL;
dataSize = 0;

```

```

dataAddress = NULL;

/* do dciGetData with the key got from dciOpen */
status = dciGetData(key, (DCIMetricId *)NULL, 0, &retBuffer, 0,
                    &dataAddress, &dataSize, &timx);
if( status != DCI_SUCCESS ) /* if FAILURE */
{
    console_printf("dciGetData FAILED. status 0x%x\n", status);

    status = dciClose(key);
    console_printf("dciClose status 0x%x\n", status);

    status = dciTerminate();
    console_printf("dciTerminate status = 0x%x\n", status);

    console_printf("EXITING #2 main\n");
    exit(-1);
}
console_printf("dciGetData PASSED. status 0x%x summary status 0x%x\n ",
               status, retBuffer->sumstatus);
if( retBuffer->sumstatus != DCI_SUCCESS )
    console_printf("There was atleast one FAILURE in the return buffer\n");

/* free the memory allocated by dciGetData */
status = dciFree((uint32)retBuffer, retBuffer->size);
console_printf("dciFree status 0x%x\n", status);

status = dciFree((uint32)dataAddress, dataSize);
console_printf("dciFree status 0x%x\n", status);

/* Close the key */
status = dciClose(key);
console_printf("dciClose status 0x%x\n", status);

/* Terminate the DCI services */
status = dciTerminate();
console_printf("dciTerminate status = 0x%x\n", status);
console_printf("EXITING #3:main\n");
exit(0);
}

```

dcigetdata - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

dcigetinstattributes

dcigetinstattributes - Syntax

Acquires metric instance attributes.

```
#include <dci.h>
```

```

DCIHandle      handle;
DCIMetricId    *metricIdList;
uint32         numIds;
DCIReturn      **bufferAddress;
uint32         bufferSize;
DCIStatus      ulrc;          /* Return code. */

ulrc = dciGetInstAttributes(handle, metricIdList,
                           numIds, bufferAddress, bufferSize);

```

dcigetInstAttributes Parameter - handle

handle (DCIHandle) - input

Handle that was returned from a prior [dcioOpen](#) call for prior open of classIdList or a superset of *metricIdList* . This field is optional. If set to 0 the argument will be ignored.

dcigetInstAttributes Parameter - metricIdList

metricIdList (DCIMetricId *) - input

Address of a list of metric identifiers. This field is optional. If set to 0 the argument will be ignored.

dcigetInstAttributes Parameter - numIds

numIds (uint32) - input

The number of input metric identifiers.

dcigetInstAttributes Parameter - bufferAddress

bufferAddress (DCIReturn **) - output

Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using `dciFree`.

dcigetInstAttributes Parameter - bufferSize

bufferSize (uint32) - input

Reserved, set to zero.

dciParamGetInstAttributes Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dciParamGetInstAttributes returns one of the following values:

DCI_SUCCESS	The DCIReturn structure has been written to the output buffer.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.
	Instance notification is NOT supported for volatile information like tasks and threads.
DCI_BADHANDLE	The handle provided is not currently open.

dciParamGetInstAttributes - Parameters

handle ([DCIHandle](#)) - input
Handle that was returned from a prior [dciParamOpen](#) call for prior open of classIdList or a superset of *metricIdList* . This field is optional. If set to 0 the argument will be ignored.

metricIdList ([DCIMetricId *](#)) - input
Address of a list of metric identifiers. This field is optional. If set to 0 the argument will be ignored.

numIds (uint32) - input
The number of input metric identifiers.

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using [dciParamFree](#).

bufferSize (uint32) - input
Reserved, set to zero.

ulrc ([DCIStatus](#)) - returns
Return code.

dciParamGetInstAttributes returns one of the following values:

DCI_SUCCESS	The DCIReturn structure has been written to the output buffer.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.
	Instance notification is NOT supported for volatile information like tasks and threads.
DCI_BADHANDLE	The handle provided is not currently open.

dciGetInstAttributes - Remarks

`dciGetInstAttributes` is used to retrieve the attribute structures for a list of instances. The metric identifiers can be optionally wildcarded for both class or instance. The `datumId` filed of the [DCIMetricId](#) is ignored. An optional handle may be provided for the `metricIdList` . If the handle is valid and if `metricIdList` contains wildcarded classes, then all metric classes of `metricIdList` are confirmed to be a subset of the opened metric classes represented by the handle, and any changed classes will be marked with a special information status, `DCI_CLASSES_CHANGED`, in the associated [DCIRetval](#) structure. Similarly, if there are instance wildcards and a valid handle, then each expanded instance is confirmed to be within a subset of metric identifiers represented by the handle and any newly added instances will be marked with a special informational status, `DCI_INSTANCES_CHANGES`. If the handle provided is zero then no such confirmation or status is provided. Not specifying a handle is useful if an application traversing the namespace wants to examine attributes, such as the label, without calling [dciOpen](#). A handle should be specified if attributes are being retrieved prior to data collection because it further validates the metric list prior to obtaining data for it. The `metricIdList` can be a subset of the metrics opened with the handle.

dcIGetInstAttributes expands any metric class and metric instance wildcards appearing in *metricIdList* and determines if each expanded metric class is currently registered in the metrics name space and if each expanded instance is also registered. For each expanded metric identifier a **DCIRetval** reply is created in the **DCIReturn** structure stored in the return buffer. The *metricOffset* member of each **DCIRetval** references the expanded **DCIMetricId** structure. The *dataOffset* member of each **DCIRetval** is set to an array of **DCIInstAttr** structures. The number of elements in this returned **DCIInstAttr** array is stored in the *count* member of the **DCIRetval** structure.

The summary status of all individual `DCIRetval` *status* members is stored in the *sumstatus* member of `DCIReturn`. This summary status represents the highest severity of status returned among all `DCIRetval` structures.

The summary status may be one of the following values:

```
DCI_SUCCESS           All status returned was successful.
```

DCI_FAILURE	There was at least one failure status.
-------------	--

For each `DCIRetval` structure returned, the `status` member may contain one of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
-------------	---

DCI_NOCLASS	The class identifier is not present in the name space.
-------------	--

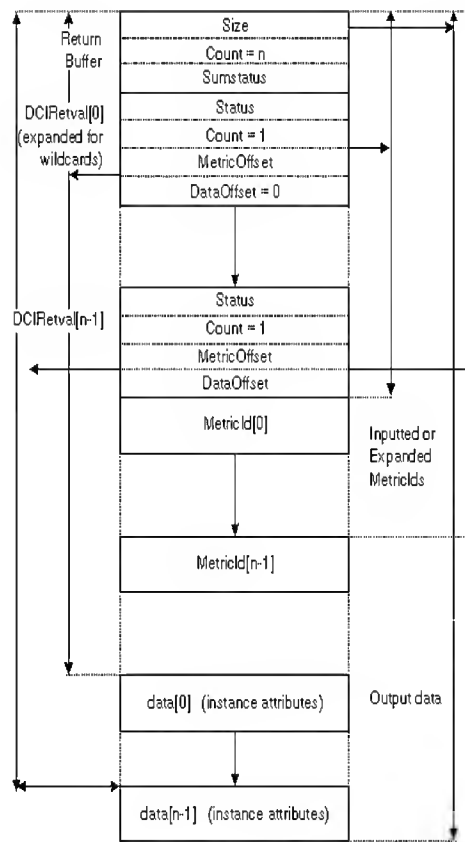
DCI_CLASSECHANGED	Classes have been added or deleted within the scope of a wildcarded class list.
-------------------	---

DCI_INSTANCECHANGED	Instances have been added or deleted within the scope of a wildcarded class list.
---------------------	---

DCI_NOINSTANCE	The requested instance identifier is not in the name space.
----------------	---

Any changes to the name space registration since the handle was created for the *metricIdList* , (volatile instances excluded), will be indicated by an appropriate status return value.

The following figure shows the return structures for `dciGetInstanceAttributes`.



dcigetInstAttributes - Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

main(int argc, char **argv, char **envp)
{
    char                *ps_version;
    DCIStatus           status;
    DCIMetricId         *metric = NULL;
    DCIClassId          *class = NULL;
    DCIInstanceId       *instance = NULL;
    DCIReturn           *retBuffer = NULL;
    uint32              *class_data = NULL;
    uint32              *inst_data = NULL;
    uint32              numIds;

    console_printf("ENTERING:main\n");

    /* Initialize dci services */
    status = dciInitialize(&ps_version);
    console_printf("dciInitialize status=0x%x\n", status);
    if(status != DCI_SUCCESS)
    {
        console_printf("dciInitialize FAILED\n");
        console_printf("EXITING #0:main\n");
        exit(-1);
    }
    console_printf("VERSION=%s\n", ps_version);

    /* Prepare the metric id structure */
    metric = (DCIMetricId *)malloc(72);
    metric->size = 76;
    metric->classId.offset = 16;
```



```

metric->instanceId.offset = 48;
metric->datumId = 255;

class = (DCIClassId *)&(metric->data);
class->size = 32;
class->identifier.offset = 16;
class->identifier.count = 4;
class->identifier.size = 4;
class_data = (uint32 *)&(class->data);
*class_data++ = 254;
*class_data++ = 1;
*class_data++ = 3;
*class_data = 1;

instance = (DCIInstanceId *)((char *)class + class->size);
instance->size = 28;
instance->levelWild = 0x3;
instance->instances.offset = 20;
instance->instances.size = 8;
instance->instances.count = 1;
inst_data = (uint32 *)&(instance->data);
*inst_data = 0;

/* Call dciGetInstAttributes */
retBuffer = NULL;
numIds = 1;
status = dciGetInstAttributes(0, metric, numIds, &retBuffer, 0);
free(metric);
if(status != DCI_SUCCESS) /* if FAILURE */
{
    console_printf("dciGetInstAttributes FAILED. status 0x%x\n",status);
    status = dciTerminate();
    console_printf("dciTerminate status = 0x%x\n",status);
    console_printf("EXITING #1 main\n");
    exit(-1);
}
console_printf("dciGetInstAttributes PASSED. status 0x%x - summary status 0x%x\n",
               status, retBuffer->sumstatus);
if( retBuffer->sumstatus != DCI_SUCCESS )
    console_printf("There was atleast one FAILURE in the return buffer\n");

/* Close the key, terminate the DCI services and Clean up */
status = dciFree((uint32)retBuffer, retBuffer->size);
console_printf("dciFree status 0x%x\n",status);
status = dciTerminate();
console_printf("dciTerminate status = 0x%x\n",status);
console_printf("EXITING #2:main\n");
exit(0);
}

```

dcigetInstAttributes - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

dcilistClassId

dcilistClassId - Syntax

Looks up a list of metric class identifiers in the metrics name space.

```
#include <dcI.h>

DCIHandle      handle;
DCIClassId     *classIdList;
uint32         numIds;
DCIReturn      **bufferAddress;
uint32         bufferSize;
DCIStatus      ulrc;          /* Return code. */

ulrc = dciListClassId(handle, classIdList,
                      numIds, bufferAddress, bufferSize);
```

dcIListClassId Parameter - handle

handle (DCIHandle) - input

The handle returned from dciOpen. This field is optional. If set to 0 the argument will be ignored.

dcIListClassId Parameter - classIdList

classIdList (DCIClassId *) - input

Address of a list of metric class identifiers. This field is optional. If set to 0 the argument will be ignored.

dcIListClassId Parameter - numIds

numIds (uint32) - input

The number of input metric class identifiers.

dcIListClassId Parameter - bufferAddress

bufferAddress (DCIReturn **) - output

Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dciFree.

dcIListClassId Parameter - bufferSize

bufferSize (uint32) - input
Reserved, set to zero.

dciParamClassId Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dciParamClassId returns one of the following values:

- DCI_SUCCESS
The DCIReturn structure has been written to the output buffer.
- DCI_NOTINITIALIZED
The requester has not yet initialized the DCI.
- DCI_SYSERROR
A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
- DCI_INVALIDARG
One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.
- DCI_BADHANDLE
The handle provided is not currently open.

dciParamClassId - Parameters

handle (DCIHandle) - input
The handle returned from dciParamOpen. This field is optional. If set to 0 the argument will be ignored.

classIdList ([DCIClassId *](#)) - input
Address of a list of metric class identifiers. This field is optional. If set to 0 the argument will be ignored.

numIds (uint32) - input
The number of input metric class identifiers.

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dciParamFree.

bufferSize (uint32) - input
Reserved, set to zero.

ulrc ([DCIStatus](#)) - returns
Return code.

dciParamClassId returns one of the following values:

- DCI_SUCCESS
The DCIReturn structure has been written to the output buffer.
- DCI_NOTINITIALIZED
The requester has not yet initialized the DCI.
- DCI_SYSERROR
A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
- DCI_INVALIDARG
One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.
- DCI_BADHANDLE
The handle provided is not currently open.

dciParamClassId - Remarks

dciParamClassId looks up a list of metric class identifiers in the metrics name space. The datumId field of the DCIMetricId is ignored. The metric class identifiers can be optionally wildcarded. dciParamClassId expands any metric class wildcards appearing in classIdList and determines if each expanded metric class is currently registered in the metrics name space.

For each expanded metric class a DCIRetval reply is created in the DCIReturn structure. The metricOffset member of each DCIRetval references the expanded DCIParmClassId structure. The dataOffset member of each DCIRetval is set to zero.

An optional handle may be provided for the classIdList. If the handle is valid and if classIdList contains wildcarded classes, then all metric classes of classIdList are confirmed to be a subset of the opened metric classes represented by the handle and any changed classes will be marked with a DCI_CLASSES_CHANGED in the associated DCIRetval structure. The classIdList is a subset of the opened metrics associated with handle. If the handle provided is zero then no such confirmation or status is provided.

The dciParamClassId command is used by a consumer to either validate a list of inputted metrics, or using wildcard values for the Class Ids, to discover what classes are available in the name space. This validated class list can then be used as input to dciParamOpen, dciParamGetClassAttr, and dciParamGetData. If the dciParamClassId command is issued after dciParamOpen then the status field on the RetVal will register the change status of non-volatile metrics.

The summary status of all individual DCIRetval structure status members is stored in the DCIReturn structure status member. This summary status represents the highest severity of status returned among all DCIRetval structures.

The summary status may be one of the following values:

DCI_SUCCESS All status returned was successful.

DCI_FAILURE There was at least one failure status.

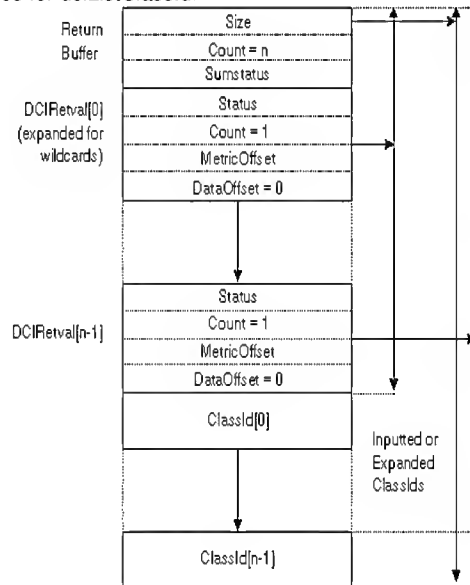
For each DCIRetval structure returned, the status member may contain one of the following values:

DCI_SUCCESS The request succeeded and there may be associated data.

DCI_NOCLASS The Class is not present in the name space.

DCI_CLASSCHANGED This new class has been added within the scope of a wildcarded class request.

The following figure shows the return structures for dciParamClassId.



dciParamClassId - Example Code

```
#include <stdio.h>
#include <stdlib.h>
```

```

#include <mk.h>
#include "dci.h"

main(int argc, char **argv, char **envp)
{
    char          *ps_version;
    DCIStatus      status;
    DCIHandle      key = 0;
    DCIReturn      *retBuffer = NULL;
    DCIClassId     *class = NULL;
    uint32         numIds = 1;
    uint32         RetBufSize = 0;
    uint32         *class_data;

    console_printf("ENTERING:main\n");

    /* Initialize dci services */
    status = dciInitialize(&ps_version);
    console_printf("dciInitialize status=0x%x\n",status);
    if(status != DCI_SUCCESS)
    {
        console_printf("dciInitialize FAILED\n");
        console_printf("EXITING #0:main\n");
        exit(-1);
    }
    console_printf("VERSION=%s\n",ps_version);

    class = (DCIClassId *)malloc(32);
    class->size = 32;
    class->identifier.offset = 16;
    class->identifier.count = 4;
    class->identifier.size = 4;
    class_data = (uint32 *)&(class->data);
    *class_data++ = 254;
    *class_data++ = 1;
    *class_data++ = 3;
    *class_data = -1;

    key = 0;
    numIds = 1;
    status = dciListClassId(key, class, numIds, &retBuffer, RetBufSize);
    free(class);
    if(status != DCI_SUCCESS) /* if FAILURE */
    {
        console_printf("dciListClassId FAILED. status 0x%x\n",status);
        status = dciTerminate();
        console_printf("dciTerminate status = 0x%x\n",status);
        console_printf("EXITING #1 main\n");
        exit(-1);
    }
    console_printf("dciListClassId PASSED. status 0x%x-summary status 0x%x\n",
                  status, retBuffer->sumstatus);
    if( retBuffer->sumstatus != DCI_SUCCESS )
        console_printf("There was atleast one FAILURE in the return buffer\n");

    /* Terminate the DCI services and Clean up */
    status = dciFree((uint32)retBuffer, retBuffer->size);
    console_printf("dciFree status 0x%x\n",status);
    status = dciTerminate();
    console_printf("dciTerminate status = 0x%x\n",status);
    console_printf("EXITING #2:main\n");
    exit(0);
}

```

dciListClassId - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

dcilistInstanceId

dcilistInstanceId - Syntax

Looks up a list of Instance identifiers in the metrics name space.

```
#include <dcil.h>

DCIHandle      handle;
DCIMetricId    *metricIdList;
uint32         numIds;
DCIReturn      **bufferAddress;
uint32         bufferSize;
DCIStatus      ulrc;          /* Return code. */

ulrc = dcilistInstanceId(handle, metricIdList,
                        numIds, bufferAddress, bufferSize);
```

dcilistInstanceId Parameter - handle

handle (DCIHandle) - input
The handle returned from dcilOpen. This field is optional. If set to 0 the argument will be ignored.

dcilistInstanceId Parameter - metricIdList

metricIdList (DCIMetricId *) - input
Address of a list of metric class and instance identifiers. This field is optional. If set to 0 the argument will be ignored.

dcilistInstanceId Parameter - numIds

numIds (uint32) - input
The number of input metric class identifiers.

dcilistInstanceId Parameter - bufferAddress

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using `dcifree`.

dcilistinstanceld Parameter - bufferSize

bufferSize (uint32) - input
Reserved, set to zero.

dcilistinstanceld Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

`dcilistinstanceld` returns one of the following values:

`DCI_SUCCESS`

The `DCIReturn` structure has been written to the output buffer.

`DCI_NOTINITIALIZED`

The requester has not yet initialized the DCI.

`DCI_SYSEERROR`

An internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

`DCI_INVALIDARG`

One of the input arguments is invalid. For example, a negative value was used for `numIds` or `bufferSize`. Also, either `metricIdList` could not be read or buffer could not be written.

`DCI_BADHANDLE`

The handle provided is not currently open.

dcilistinstanceld - Parameters

handle ([DCIHandle](#)) - input
The handle returned from `dcioopen`. This field is optional. If set to 0 the argument will be ignored.

metricIdList ([DCIMetricId *](#)) - input
Address of a list of metric class and instance identifiers. This field is optional. If set to 0 the argument will be ignored.

numIds (uint32) - input
The number of input metric class identifiers.

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using `dcifree`.

bufferSize (uint32) - input
Reserved, set to zero.

ulrc ([DCIStatus](#)) - returns
Return code.

`dcilistinstanceld` returns one of the following values:

DCI_SUCCESS	The DCIReturn structure has been written to the output buffer.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.
DCI_BADHANDLE	The handle provided is not currently open.

dciParamInstanceId - Remarks

dciParamInstance routine looks up a list of instance identifiers in the metrics name space and loads the methods used to obtain this information if it is not already loaded. The metric class and instance identifiers can be optionally wildcarded. The datumId field of the DCIMetricId is ignored. dciParamInstance determines if each class/instance pair in metricIdList is currently registered in the metrics name space, expands any metric class or instance wildcards, and determines if each expanded metric class is currently registered in the metrics name space and if each expanded instance is also registered. For each expanded metric identifier a DCIRetval reply is created in the DCIReturn structure stored in the return buffer.

dciParamInstance is used by a consumer to either validate a list of inputted metrics, or using wildcard values for the Class and Instance Ids, to discover what metrics are available in the name space. This validated metric list can then be used as input to dciParamOpen, dciParamGetInstanceAttr, and dciParamGetData. If the dciParamInstance command is issued after dciParamOpen with the handle obtained from dciParamOpen then the status field on the RetVal will register the change status of non-volatile metrics.

Note: There is no notification of changes for volatile instances.

An optional handle may be provided for the instanceIdList. If the handle is valid and if instanceIdList contains wildcarded classes, then all metric classes of instanceIdList are confirmed to be a subset of the opened metric classes represented by the handle, and any changed instances will be marked with a DCI_INSTANCECHANGED in the associated DCIRetval structure. The instanceIdList is a subset of the opened metrics associated with handle. If the handle provided is 0 then no such confirmation or status is provided.

The summary status of all individual DCIRetval structure status members is stored in the DCIReturn structure status member. This summary status represents the highest severity of status returned among all DCIRetval structures.

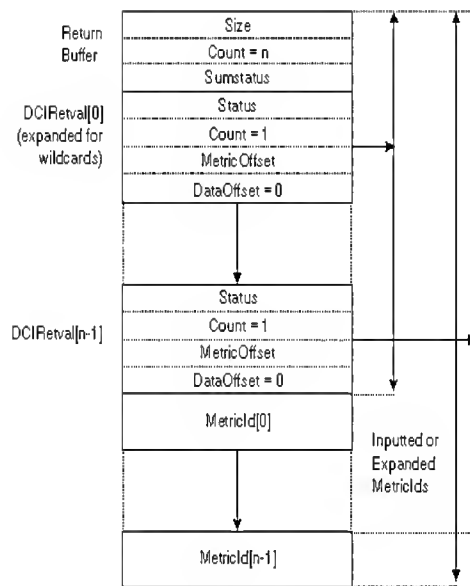
The summary status may be one of the following values:

DCI_SUCCESS	All status returned was successful.
DCI_FAILURE	There was at least one failure status.

For each DCIRetval structure returned, the status member may contain one of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
DCI_NOCLASS	The Class is not present in the name space.
DCI_NOINSTANCE	The requested instance identifier is not in the name space.
DCI_INSTANCECHANGED	This new instance has been added within the scope of a wildcarded instance request.

The following figure shows the return structures for dciParamInstance.



dciParamInstanceId - Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

main(int argc, char **argv, char **envp)
{
    char                *ps_version;
    DCIStatus           status;
    DCIMetricId         *metric = NULL;
    DCIClassId          *class = NULL;
    DCIInstanceId       *instance = NULL;
    DCIReturn           *retBuffer = NULL;
    uint32              *class_data = NULL;
    uint32              *inst_data = NULL;
    uint32              numIds;

    console_printf("ENTERING:main\n");

    /* Initialize dci services */
    status = dciInitialize(&ps_version);
    console_printf("dciInitialize status=0x%x\n", status);
    if(status != DCI_SUCCESS)
    {
        console_printf("dciInitialize FAILED\n");
        console_printf("EXITING #0:main\n");
        exit(-1);
    }
    console_printf("VERSION=%s\n", ps_version);

    /* Prepare the metric id structure */
    metric = (DCIMetricId *)malloc(72);
    metric->size = 72;
    metric->classId.offset = 16;
    metric->instanceId.offset = 48;
    metric->datumId = 255;

    class = (DCIClassId *)&(metric->data);
    class->size = 32;
    class->identifier.offset = 16;
    class->identifier.count = 4;
    class->identifier.size = 4;
    class_data = (uint32 *)&(class->data);
    *class_data++ = 254;
    *class_data++ = 1;
}
```

```

*class_data++ = 2;
*class_data    = 1;

instance = (DCIInstanceId *) ((char *)class + class->size);
instance->size = 24;
instance->levelWild = 0x1;
instance->instances.offset = 20;
instance->instances.size = 4;
instance->instances.count = 1;
inst_data = (uint32 *)&(instance->data);
*inst_data = 0;

/* Call dciListInstanceId */
retBuffer = NULL;
numIds = 1;
status = dciListInstanceId(0, metric, numIds, &retBuffer, 0);
free(metric);
if(status != DCI_SUCCESS) /* if FAILURE */
{
    console_printf("dciListInstanceId FAILED. status 0x%x\n",status);
    status = dciTerminate();
    console_printf("dciTerminate status = 0x%x\n",status);
    console_printf("EXITING #1 main\n");
    exit(-1);
}
console_printf("dciListInstanceId PASSED. status 0x%x - summary status 0x%x\n", status, retBuffer->sumstatus);
if( retBuffer->sumstatus != DCI_SUCCESS )
    console_printf("There was atleast one FAILURE in the return buffer\n");

/* Close the key, terminate the DCI services and Clean up */
status = dciFree((uint32)retBuffer, retBuffer->size);
console_printf("dciFree status 0x%x\n",status);
status = dciTerminate();
console_printf("dciTerminate status = 0x%x\n",status);
console_printf("EXITING #2:main\n");
exit(0);
}

```

dcilistInstanceId - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

dcioPen

dcioPen - Syntax

Opens a list of metrics and returns an access handle.

```
#include <dcI.h>
```

DCIHandleID	<i>*handle;</i>
DCIMetricId	<i>*metricIdList;</i>
uint32	<i>numIds;</i>

```

DCIReturn      **bufferAddress;
uint32         bufferSize;
uint32         handleFlags;
DCIStatus      ulrc;          /* Return code. */

ulrc = dciOpen(handle, metricIdList, numIds,
               bufferAddress, bufferSize, handleFlags);

```

dcisOpen Parameter - handle

handle (DCIHandleID *) - output
 A pointer to the location to return the DCI handle.

dcisOpen Parameter - metricIdList

metricIdList (DCIMetricId *) - input
 A pointer to a list of metric class and instance identifiers.

dcisOpen Parameter - numIds

numIds (uint32) - input
 The number of input metric class identifiers.

dcisOpen Parameter - bufferAddress

bufferAddress (DCIReturn **) - output
 A pointer to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dciFree.

dcisOpen Parameter - bufferSize

bufferSize (uint32) - output
 Reserved, set to zero.

dcisOpen Parameter - handleFlags

handleFlags (uint32) - output
Reserved, set to zero.

dcisOpen Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dciTerninate returns one of the following values:

DCI_SUCCESS

The DCIReturn structure has been written to the output buffer.

DCI_SYSERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_NOTINITIALIZED

The DCI subsystem is not currently initialized.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.

dcisOpen - Parameters

handle ([DCIHandleID *](#)) - output
A pointer to the location to return the DCI handle.

metricIdList ([DCIMetricId *](#)) - input
A pointer to a list of metric class and instance identifiers.

numIds (uint32) - input
The number of input metric class identifiers.

bufferAddress ([DCIReturn **](#)) - output
A pointer to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dciFree.

bufferSize (uint32) - output
Reserved, set to zero.

handleFlags (uint32) - output
Reserved, set to zero.

ulrc ([DCIStatus](#)) - returns
Return code.

dciTerninate returns one of the following values:

DCI_SUCCESS

The DCIReturn structure has been written to the output buffer.

DCI_SYSERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_NOTINITIALIZED

The DCI subsystem is not currently initialized.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.

dciParam - Remarks

dciParam loads the appropriate libraries and/or the Performance Server for the metrics chosen.

dciParam instructs the Performance Services to perform an access check for every metric in the metricIdList.

Note: If successful for at least one metric, a handle is returned which can be used in subsequent operations to access all or some of the metrics in the metricIdList.

Implementations may perform access checks only at the time of the dciParam call. The handle is valid only within the task that the dciParam is issued in. The Performance Services sets up a notify with the name space so that registration changes to the requested metrics are registered with the Performance Services for each task's open sessions for non-volatile metrics. The appropriate change notification is then given to the consumer at the time of dciParamData, dciParamClassId, dciParamInstancelId, dciParamGetClassAttr, and dciParamGetInstanceAttr for the non-volatile metric(s) involved.

Handleflags are not supported for Release 1.

dciParam performs an existence check for every class and instance in the metricIdList and writes status into the given output buffer using the standard DCI return structure. Only the status and metricOffset fields of the DCIRetval reply are used in the return structure. The dataOffset in DCIRetval is set to zero since it is unused. For each expanded metric identifier a DCIRetval reply is created in the DCIReturn structure stored in the return buffer pointed to by bufferaddress. The metricOffset member of each DCIRetval specifies a location relative to buffer address and references the expanded DCIMetricId structure.

The summary status of all individual DCIRetval structure status members is stored in the status member of the DCIReturn structure. This summary status represents the highest severity of status returned among all DCIRetval structures.

DCI_SUCCESS

All status returned was successful.

DCI_FAILURE

There was at least one failure status.

For each DCIRetval structure returned, the status member may contain the following:

DCI_SUCCESS

The request was successful and there may be associated data.

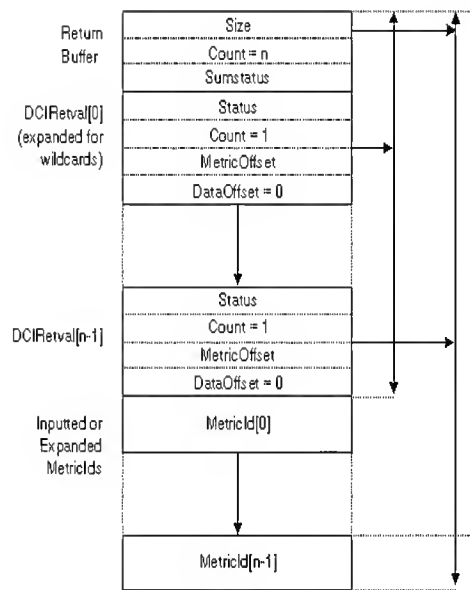
DCI_NOCLASS

The class is not present in the name space.

DCI_NOINSTANCE

The requested instance identifier is not in the name space.

The following figure shows the return structures for dciParam.



dcisOpen - Related Functions

- [dciclose](#)

dcisOpen - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

DCI Provider Functions

This sections describes the following Data Capture Interface (DCI) functions used by Provider applications:

- [dcieZRegister](#)
- [dcieZUnregister](#)
- [dciregister](#)
- [dciregister](#)
- [dciregister](#)
- [dciregister](#)
- [dciregister](#)
- [DCI_SPI_NAMEfunctionname](#)
- [DCI_SPI_RPCfunctionname](#)

dcieZRegister

dcieZRegister - Syntax

dcieZRegister issues [dciiInitialize](#) and [dciiRegister](#) for each individual classid level, and [dciiAddInstance](#) for a single instance level. dcieZRegister creates all of the necessary class, instance, label, data and instance level structures needed for a complete metricid registration based on information provided by the provider, and documented defaults. dcieZRegister issues a [dciiTerminate](#) upon completion of its registration.

```
#include <dcii.h>

char      *classidstr;
char      *classlblstr;
char      *instancelblstr;
uint32    funcaddr;
void      *userword;
uint32    datasize;
char      *datalblstr;
uint32    datatype;
uint32    dataunits;
uint32    flags;
uint32    instance;
uint32    itype;
uint32    dtype;
uint32    dunits;
uint32    datasize;
char      *ulrc;          /* MetricIdString */

ulrc = dcieZRegister(classidstr, classlblstr,
                    instancelblstr, funcaddr, userword,
                    datasize, datalblstr, datatype, dataunits,
                    flags, instance, itype, dtype, dunits,
                    datasize);
```

dcieZRegister Parameter - classidstr

classidstr (char *) - input

A pointer to a character string which contains a class id for each supported level. Class ids are separated by the first character which is treated as the delimiter. The delimiter cannot be alphanumeric because it could conflict with the assigned instance id, and this delimiter is used in the returned metricidstr. For example, /254/5/1/1.

dcieZRegister Parameter - classlblstr

classlblstr (char *) - input

A pointer to a character string which contains a class label for each supported class id. Class labels are separated by the first character which is treated as the delimiter. The delimiter cannot be alphanumeric. For example, /IBM/LEVEL1/LEVEL2/LEVEL3.

dcieZRegister Parameter - instancelblstr

instanceblstr (char *) - input

A pointer to a character string which contains a single instance label for the supported instance id. The first character is treated as a delimiter and cannot be alphanumeric. For example, /INST1.

dcieZRegister Parameter - funcaddr

funcaddr (uint32) - input

The address where the function to retrieve metric information was loaded in the provider's address space.

dcieZRegister Parameter - userword

userword - input

A pointer to a provider-supplied user word for this metric id.

dcieZRegister Parameter - datasize

datasize (uint32) - input

The size in bytes of the data returned for this data label.

dcieZRegister Parameter - datablstr

datablstr (char *) - input

A pointer to a character string which contains a datum label for each supported datum id. Datum labels are separated by the first character which is treated as the delimiter. The delimiter cannot be alphanumeric. For example, /LABEL1.

dcieZRegister Parameter - datatype

datatype (uint32) - input

The type used for the datum. This defaults to all datums whose label exists in datablstr unless the multitype flag is set. If multitype is set, then this parameter is ignored and the optional parameters after the flags field and possible instance information are checked for data type.

dcieZRegister Parameter - dataunits

dataunits (uint32) - input

The units used for the datum. This defaults to all datums whose labels exist in datablstr unless the multitype flag is set. If multitype is set, then this parameter is ignored and the optional parameters after the flags field and possible instance information are checked for data type.

dcieZRegister Parameter - flags

flags (uint32) - input

Determine the action to be taken based on the following bit values:

- | | |
|---|---|
| 1 | tid/pid/pathname requested in local extensions of the Instance Attributes structure. |
| 2 | Instance id and instance type are first optional parameters. |
| 4 | Multiple type and units datums are requested and follow in grouped pairs of type and units in the optional parameter section. |
-

dcieZRegister Parameter - instance

instance (uint32) - input

(Option if flag = instance) Unique instance id within class. Only one instance id is allowed.

dcieZRegister Parameter - itype

itype (uint32) - input

(Option if flag = instance) The instance id type of the instance level.

dcieZRegister Parameter - dtype

dtype (uint32) - input

(Option if flag = multitype) One per datum label. dtype, dunits and datasize are grouped in as many pairs as there are datum labels.

dcieZRegister Parameter - dunits

dunits (uint32) - input

(Option if flag = multitype) One per datum label. dtype, dunits and datasize are grouped in as many pairs as there are datum labels.

dcieZRegister Parameter - datasize

datasize (uint32) - input

The size in bytes of the data returned for this data label. dtype, duints and datasize are grouped in as many pairs as there are datum labels.

dcieZRegister Return Value - ulrc

ulrc (char *) - returns
MetricIdString

If all class levels and instance id's are successfully verified, dcieZRegister returns the *ulrc* that was registered in the name space. This string can subsequently be used to unregister the MetricId using [dcieZUnregister](#). If all class levels and instance id's are not successfully verified, a NULL is returned in *ulrc*.

dcieZRegister - Parameters

classidstr (char *) - input

A pointer to a character string which contains a class id for each supported level. Class ids are separated by the first character which is treated as the delimiter. The delimiter cannot be alphanumeric because it could conflict with the assigned instance id, and this delimiter is used in the returned metricidstr. For example, /254/5/1/1.

classlblstr (char *) - input

A pointer to a character string which contains a class label for each supported class id. Class labels are separated by the first character which is treated as the delimiter. The delimiter cannot be alphanumeric. For example, /IBM/LEVEL1/LEVEL2/LEVEL3.

instancelblstr (char *) - input

A pointer to a character string which contains a single instance label for the supported instance id. The first character is treated as a delimiter and cannot be alphanumeric. For example, /INST1.

funcaddr (uint32) - input

The address where the function to retrieve metric information was loaded in the provider's address space.

userword - input

A pointer to a provider-supplied user word for this metric id.

datasize (uint32) - input

The size in bytes of the data returned for this data label.

datalblstr (char *) - input

A pointer to a character string which contains a dataum label for each supported datum id. Datum labels are separated by the first character which is treated as the delimiter. The delimiter cannot be alphanumeric. For example, /LABEL1.

datatype (uint32) - input

The type used for the datum. This defaults to all datums whose label exists in datalblstr unless the multitype flag is set. If multitype is set, then this parameter is ignored and the optional parameters after the flags field and possible instance information are checked for data type.

dataunits (uint32) - input

The units used for the datum. This defaults to all datums whose labels exist in datalblstr unless the multitype flag is set. If multitype is set, then this parameter is ignored and the optional parameters after the flags field and possible instance information are checked for data type.

flags (uint32) - input

Determine the action to be taken based on the following bit values:

1

tid/pid/pathname requested in local extensions of the Instance Attributes structure.

- | | |
|---|---|
| 2 | Instance id and instance type are first optional parameters. |
| 4 | Multiple type and units datums are requested and follow in grouped pairs of type and units in the optional parameter section. |

instance (uint32) - input
 (Option if flag = instance) Unique instance id within class. Only one instance id is allowed.

itype (uint32) - input
 (Option if flag = instance) The instance id type of the instance level.

dtype (uint32) - input
 (Option if flag = multitype) One per datum label. dtype, duints and datasize are grouped in as many pairs as there are datum labels.

dunits (uint32) - input
 (Option if flag = multitype) One per datum label. dtype, duints and datasize are grouped in as many pairs as there are datum labels.

datasize (uint32) - input
 The size in bytes of the data returned for this data label. dtype, duints and datasize are grouped in as many pairs as there are datum labels.

ulrc (char *) - returns
 MetricIdString

If all class levels and instance id's are successfully verified, dciEZRegister returns the *ulrc* that was registered in the name space. This string can subsequently be used to unregister the MetricId using [dciEZUnregister](#). If all class levels and instance id's are not successfully verified, a NULL is returned in *ulrc*.

dcieZRegister - Remarks

dcieZRegister registers a single metricid and, if requested, assigns an instance id to the provided class id, sets the local extensions of the DCIInstAttribute structure to the appropriate tid/pid/pathname of the provider, and assigns multiple datums to the different data types, units and sizes provided by the provider in the dtype, duints and datasize parameters. dcieZRegister will guarantee that the instance id it selects is unique enough to successfully register the metric id. The attributes structures for class attributes, data attributes, instance attributes, instance levels, and the local extensions for instance attributes are generated using the provider-supplied parameters and the defaults given below. dciRegister and dciAddInstance are issued to properly register the class ids and instance id into the name space. dcieZRegister obtains storage and returns a pointer to this storage as its function's return. dcieZRegister returns the metricidstr in the storage if the registration was successful; otherwise, it returns a NULL pointer. It is up to the issuer to return any storage malloc'd via free when a non-NULL pointer is returned.

The default values set by dcieZRegister are:

- Class attribute flags default to DCI_ENABLED and non-persistent.
- All access and event attributes are not supported for Release 1 and their offsets are set to zero.
- Class attribute local extensions are not supported and their offset is set to zero.
- DCI_SPR_RPC is the only method type supported by dcieZRegister.
- For method type DCI_SPI_RPC the only call type supported is DCI_GETDATA. This is the default value.
- The default data attribute flag is DCI_QUERYABLE.
- Data attribute datum id is set to the position in the label string.
- The default instance attribute flag is zero and non-persistent.
- Instance attribute local extensions are set to tid/pid/pathname if requested.

dcieZRegister - Example Code

```

#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

DCIStatus counter_api();
int counter_userword;

main()
{
    DCIStatus      status;
    uint32         stringAddress = 0;
    char           *dci_metricString;

    /* Register a metric */
    stringAddress=dciEZRegister("/254/5/1/1",          /* classidstr */
                                "|NON_UMA|IBM_SAMPLE|COUNTER|SHARED_MEM_CNT", /*class label str*/
                                "|SAMPLECOUNTER",      /* inst Label str */
                                (uint32)counter_api,    /* func address */
                                &counter_userword,     /* userword */
                                4,                     /* data Size */
                                "/counter",            /* data Labelstr */
                                UMA_UINT32,            /* data type */
                                DCI_COUNT,             /* data Units */
                                0);                    /* flags */
    if (stringAddress == 0)
    {
        console_printf("Registration of metrics did not complete properly\n");
        console_printf("EXITING #0:main\n");
        return(-1);
    }

    /* print out the string just to see if we got it ok */
    dci_metricString=(char *)stringAddress;
    console_printf("METRIC_ID STRING=%s\n",dci_metricString);
    console_printf("Registration completed. Now going to unregister.....\n");

    /* Unregister the metric */
    status=dciEZUnregister(stringAddress);
    console_printf("dciEZUnregister status=0x%x\n",status);
    console_printf("EXITING #1:main\n");
    exit(0);
}

DCIStatus counter_api(
    char      *metricidstr,
    void      *retBuffer,
    void      *userword,
    uint32    calltype,
    uint32    datasize
)
{
}

```

dcieZRegister - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

dcieZUnregister

dcieZUnregister - Syntax

Unregisters a metricid.

```
#include <dcie.h>

char      *metricidstr;
DCIStatus  ulrc;      /* Return code. */

ulrc = dcieZUnregister(metricidstr);
```

dcieZUnregister Parameter - metricidstr

metricidstr (char *) - input

A pointer to a character string which contains a classid for each registered class, with the final entry being an instance id. The first character given is treated as the delimiter used to separate classids, the last classid and the instance id within the string. The delimiter given cannot be alphanumeric. For example, /254/5/1/1/0. This metricidstr is the string returned by dcieZRegister for classid 254/5/1/1 with an instance id of 0.

dcieZUnregister Return Value - ulrc

ulrc (DCIStatus) - returns

Return code.

dcieZUnregister returns one of the following values:

DCI_SUCCESS

The operation completed successfully. All requested classids have been removed from the name space and the DCIReturn structure has been written to the output buffer.

DCI_INVALIDARG

One of the input arguments is invalid. For example, the provided string does not contain a valid classid or instance id.

DCI_SYSEERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

dcieZUnregister - Parameters

metricidstr (char *) - input

A pointer to a character string which contains a classid for each registered class, with the final entry being an instance id. The first character given is treated as the delimiter used to separate classids, the last classid and the instance id within the string. The delimiter given cannot be alphanumeric. For example, /254/5/1/1/0. This metricidstr is the string returned by dcieZRegister for classid 254/5/1/1 with an instance id of 0.

ulrc (DCIStatus) - returns

Return code.

dcieZUnregister returns one of the following values:

DCI_SUCCESS

	The operation completed successfully. All requested classids have been removed from the name space and the DCIReturn structure has been written to the output buffer.
DCI_INVALIDARG	One of the input arguments is invalid. For example, the provided string does not contain a valid classid or instance id.
DCI_SYSERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

dcieZUnregister - Remarks

dcieZUnregister allows providers to remove one their metricids from the name space. This routine takes as input a character string for a metricid in which the first character is the delimiter used between classids and instanceids. If there are multiple instances registered under the provided classid, then only the provided instance id will be unregistered from the name space. The classid is not removed until after the last instance id is removed from the name space.

dcieZUnregister - Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mk.h>
#include "dci.h"

DCIStatus counter_api();
int counter_userword;

main()
{
    DCIStatus      status;
    uint32         stringAddress = 0;
    char           *dci_metricString;

    /* Register a metric */
    stringAddress=dcieZRegister("/254/5/1/1",           /* classidstr */
                               "|NON_UMA|IBM_SAMPLE|COUNTER|SHARED_MEM_CNT", /*class label str*/
                               "|SAMPLECOUNTER",        /* inst Label str */
                               (uint32)counter_api,      /* func address */
                               &counter_userword,       /* userword */
                               4,                       /* data Size */
                               "/counter",              /* data Labelstr */
                               UMA_UINT32,              /* data type */
                               DCI_COUNT,               /* data Units */
                               0);                      /* flags */
    if (stringAddress == 0)
    {
        console_printf("Registration of metrics did not complete properly\n");
        console_printf("EXITING #0:main\n");
        return(-1);
    }

    /* print out the string just to see if we got it ok */
    dci_metricString=(char *)stringAddress;
    console_printf("METRIC_ID STRING=%s\n",dci_metricString);
    console_printf("Registration completed. Now going to unregister.....\n");

    /* Unregister the metric */
    status=dcieZUnregister(stringAddress);
    console_printf("dcieZUnregister status=0x%x\n",status);
    console_printf("EXITING #1:main\n");
    exit(0);
}

DCIStatus counter_api(
    char      *metricidstr,
    void      *retBuffer,
    void      *userword,
    uint32    calltype,
    uint32    datasize
```

```
{
    )
}
```

dciEZUnregister - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

dciRegister

dciRegister - Syntax

Registers a list of metric class identifiers.

```
#include <dci.h>

DCIClassId      *classIdList;
DCIClassAttributes *classAttrList;
uint32          numIds;
DCIReturn       **bufferAddress;
uint32          bufferSize;
DCIStatus        ulrc;          /* Return code. */

ulrc = dciRegister(classIdList, classAttrList,
                  numIds, bufferAddress, bufferSize);
```

dciRegister Parameter - classIdList

classIdList ([DCIClassId](#) *) - input
Address of a list of metric class identifiers.

dciRegister Parameter - classAttrList

classAttrList ([DCIClassAttributes](#) *) - input
Address of a list of metric class attributes . structures.

dcRegister Parameter - numIds

numIds (uint32) - input
The number of input metric classes to register.

dcRegister Parameter - bufferAddress

bufferAddress (DCIReturn **) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dcFree.

dcRegister Parameter - bufferSize

bufferSize (uint32) - input
Reserved, set to zero.

dcRegister Return Value - ulrc

ulrc (DCIStatus) - returns
Return code.

dcRegister returns one of the following values:

- DCI_SUCCESS
The DCIReturn structure has been written to the output buffer.
 - DCI_SYSERROR
A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
 - DCI_INVALIDARG
One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricAttrList could not be read or buffer could not be written.
 - DCI_NOTINITIALIZED
The DCI subsystem is not currently initialized.
-

dcRegister - Parameters

classIdList (DCIClassId *) - input
Address of a list of metric class identifiers.

classAttrList (DCIClassAttributes *) - input
Address of a list of metric class attributes . structures.

numIds (uint32) - input
The number of input metric classes to register.

bufferAddress (DCIReturn **) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dciFree.

bufferSize (uint32) - input
Reserved, set to zero.

ulrc (DCIStatus) - returns
Return code.

dciRegister returns one of the following values:

DCI_SUCCESS	The DCIReturn structure has been written to the output buffer.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricAttrList could not be read or buffer could not be written.
DCI_NOTINITIALIZED	The DCI subsystem is not currently initialized.

dciRegister - Remarks

dciRegister registers a list of classids within the name space and allows providers to specify basic attribute structures for lists of classids. For each DCIClassId element in classIdList, there is a corresponding DCIClassAttr structure which establishes the definition of the new class. Information such as the number of instance levels and their types, as well as the label and type of all datum metrics which are provided. This routine takes as input a buffer with an array of DCIClassAttriributes structures and a count of the number of inputted classids. The classIdList field of each attribute structure must be completely specified. This routine cannot be used with a wildcard values. The return address, *bufferAddress, must be NULL. The caller is then responsible for subsequently freeing the allocated memory by using dciFree.

The classid list given for a dciRegister must be in ascending order with the full classid being the last one registered, whereas the classid list given for dciUnregister must be in descending order with the full classid being the first one unregistered. This is due to the binary tree implementation for Performance Services Release 1.

Note: Upon return dciRegister will fill the return buffer specified by bufferAddress with a DCIReturn structure. For each input DCIClassId, there is a corresponding DCIRetval structure produced in the return buffer. The metricOffset field of the DCIRetval specifies an offset from the buffer specified by bufferAddress and contains the DCIClassId associated with the status field of the DCIRetval structure. The dataOffset field is set to zero.

The summary status of all individual DCIRetval structure status members is stored in the DCIReturn structure status member. This summary status represents the highest severity of status returned among all DCIRetval structures.

DCI_SUCCESS	All status returned was successful.
-------------	-------------------------------------

DCI_FAILURE	There was at least one failure status.
-------------	--

For each DCIRetval structure returned, the status member may contain one of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
-------------	---

DCI_NOCLASS	The class identifier is not present in the name space.
-------------	--

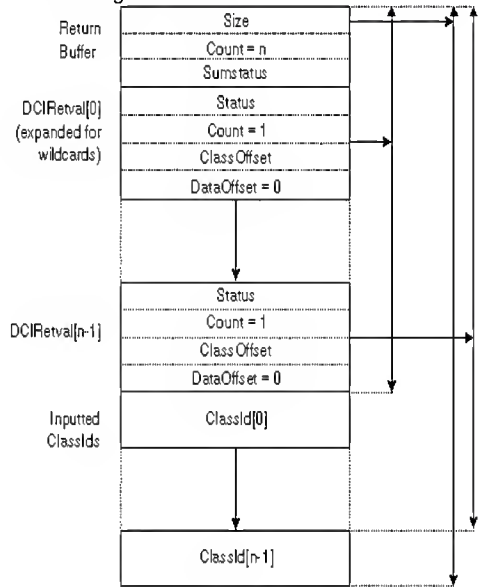
DCI_NOWILDCARD	A wildcard cannot be used in this context.
----------------	--

DCI_CLASSEXISTS	A class of this level already exists in the name space.
-----------------	---

DCI_CLASSNOTPERSISTENT
If the parent class is not persistent, and the new DCIClassAttr specifies persistence.

DCI_INVALIDFIELD
The associated DCIClassAttr structure has an invalid field.

The following figure shows the return structures for dciRegister.



dciRegister - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

dciUnregister

dciUnregister - Syntax

Unregisters a list of metrics

```
#include <dci.h>

DCIClassId    *classIdList;
uint32        numIds;
DCIReturn     **bufferAddress;
uint32        bufferSize;
DCIStatus     ulrc;          /* Return code. */

ulrc = dciUnregister(classIdList, numIds,
                    bufferAddress, bufferSize);
```

dciUnregister Parameter - classIdList

classIdList ([DCIClassId *](#)) - input
Address of a list of class identifiers to unregister.

dciUnregister Parameter - numIds

numIds (uint32) - input
The number of input metric classes to unregister.

dciUnregister Parameter - bufferAddress

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to zero so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dciFree.

dciUnregister Parameter - bufferSize

bufferSize (uint32) - output
Reserved, set to zero.

dciUnregister Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dciUnregister returns one of the following values:

- DCI_SUCCESS
The DCIReturn structure has been written to the output buffer.
- DCI_NOTINITIALIZED
The DCI subsystem is not currently initialized.
- DCI_INVALIDARG
One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.
- DCI_SYSEERROR
A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricAttrList could not be read or buffer could not be written.

dcUnregister - Parameters

classIdList (DCIClassId *) - input

Address of a list of class identifiers to unregister.

numIds (uint32) - input

The number of input metric classes to unregister.

bufferAddress (DCIReturn **) - output

Points to the address of a return value buffer.

Must be set to zero so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dcFree.

bufferSize (uint32) - output

Reserved, set to zero.

ulrc (DCIStatus) - returns

Return code.

dcUnregister returns one of the following values:

DCI_SUCCESS

The DCIReturn structure has been written to the output buffer.

DCI_NOTINITIALIZED

The DCI subsystem is not currently initialized.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.

DCI_SYSEERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricAttrList could not be read or buffer could not be written.

dcUnregister - Remarks

dcUnregister allows providers to remove one or more of their registered classes from the namespace.

Note: It is not possible to delete a name space class level if it still has any instances or lower level classid existing.

dcUnregister takes as input a buffer with an array of DCIClassId structures and a count of the number of inputted classids. Each DCIClassId may include the class identifier wildcard value, DCI_ALL. dcUnregister determines if each classid in the classIdList is currently registered in the metrics name space, expands any class wildcards in the classOffset buffer to registered classids, and unregisters each classid from the name space. For each expanded metric class a DCIRetval references the expanded DCIClassId structure and the dataOffset member is set to zero and the data buffers are not used. The caller of this routine must provide a NULL BufferAddress pointer. The caller is then responsible for subsequently freeing the allocated memory by using dcFree. Upon return dcUnregister will fill the buffer with an array of return structures for each requested classid.

It is not possible to delete a name space level if it still has any instances or lower level classid existing, or if the class is of type DCI_PERSISTENT.

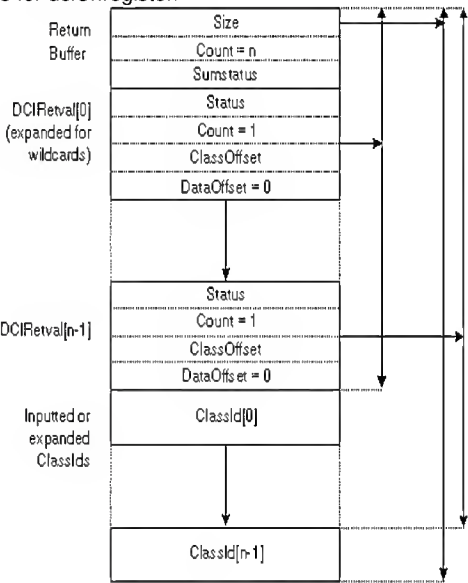
Note: The classid list given for a dcRegister must be in ascending order with the full classid being the last one registered, whereas the classid list given for dcUnregister must be in descending order with the full classid being the first one unregistered. This is due to the binary tree implementation for Performance Services Release 1.

The summary status of all individual DCIRetval structure status members is stored in the DCIReturn structure status member. This summary status represents the highest severity of status returned among all DCIRetval structures.

The summary status may be one of the following values:

DCI_SUCCESS	All status returned was successful.
DCI_FAILURE	There was at least one failure status.
For each DCIRetval structure returned, the status member may contain one of the following values:	
DCI_SUCCESS	The Operation was a success.
DCI_NOCLASS	The requested metric class identifier is not present in the name space.
DCI_CLASSNOTEMPTY	The class could not be removed because either there are subclasses still registered or instance still defined for this class.

The following figure shows the return structures for dciUnregister.



dciUnregister - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)

dciAddInstance

dciAddInstance - Syntax

Adds an instance or list of instances to a metric.

```
#include <dc_i.h>

DCIClassId      *classId;
uint32          numIds;
DCIInstanceId   *instanceIdList;
DCIInstAttr     *instAttrList;
DCIMethod       methodList;
DCIReturn       **bufferAddress;
uint32          bufferSize;
DCIStatus       ulrc;          /* Return code. */

ulrc = dciAddInstance(classId, numIds, instanceIdList,
                      instAttrList, methodList, bufferAddress,
                      bufferSize);
```

dc_iAddInstance Parameter - classId

classId ([DCIClassId](#) *) - input
Address of a single classId to add instances to. This may not contain any wildcards.

dc_iAddInstance Parameter - numIds

numIds (uint32) - input
The number of a list of class identifiers.

dc_iAddInstance Parameter - instanceIdList

instanceIdList ([DCIInstanceId](#) *) - input
Address of a list of instance identifiers to be added to the class. This may not contain any wildcards.

dc_iAddInstance Parameter - instAttrList

instAttrList ([DCIInstAttr](#) *) - input
Address of a list of instance attributes with a one-to-one correspondence to the list of instance identifiers being added.

dc_iAddInstance Parameter - methodList

methodList ([DCIMethod](#)) - input
Address of a list of instance method structures with a one-to-one correspondence to the list of instance identifiers being added. If this is zero, no instance methods will be added and there must be a class method already registered for the given class. Either all instances in

this list have methods or none do.

dciParamAddInstance Parameter - bufferAddress

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using `dciParamFree`.

dciParamAddInstance Parameter - bufferSize

bufferSize ([uint32](#)) - input
Reserved, set to zero.

dciParamAddInstance Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

`dciParamAddInstance` returns one of the following values:

<code>DCI_SUCCESS</code>	The <code>DCIReturn</code> structure has been written to the output buffer.
<code>DCI_SYSERROR</code>	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
<code>DCI_NOTINITIALIZED</code>	The DCI subsystem is not currently initialized.
<code>DCI_INVALIDARG</code>	One of the input arguments is invalid. For example, a negative value was used for <i>numIds</i> or <i>bufferSize</i> , the <i>methodList</i> could not be read, or data could not be written to <i>bufferAddress</i> .

dciParamAddInstance - Parameters

classId ([DCIClassId *](#)) - input
Address of a single classId to add instances to. This may not contain any wildcards.

numIds ([uint32](#)) - input
The number of a list of class identifiers.

instanceIdList ([DCIInstanceId *](#)) - input
Address of a list of instance identifiers to be added to the class. This may not contain any wildcards.

instAttrList ([DCIInstAttr *](#)) - input
Address of a list of instance attributes with a one-to-one correspondence to the list of instance identifiers being added.

methodList ([DCIMethod](#)) - input

Address of a list of instance method structures with a one-to-one correspondence to the list of instance identifiers being added. If this is zero, no instance methods will be added and there must be a class method already registered for the given class. Either all instances in this list have methods or none do.

bufferAddress ([DCIReturn **](#)) - output
Points to the address of a return value buffer.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using `dcifree`.

bufferSize (uint32) - input
Reserved, set to zero.

ulrc ([DCIStatus](#)) - returns
Return code.

`dcAddInstance` returns one of the following values:

DCI_SUCCESS	The DCIReturn structure has been written to the output buffer.
DCI_SYSEERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_NOTINITIALIZED	The DCI subsystem is not currently initialized.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for <i>numlds</i> or <i>bufferSize</i> , the <i>methodList</i> could not be read, or data could not be written to <i>bufferAddress</i> .

dcAddInstance - Remarks

`dcAddInstance` adds a list of instances to a class. This routine takes as input an array of instance identifiers, arrays of attributes, and optionally methods that must correspond one-to-one with the array of instance identifiers. A method in an instance attribute overrides any method found in a class attribute.

The instance attributes specified in *instanceIdList* can be retrieved by a consumer using [dcGetInstAttributes](#). The *methodList* allows a provider to specify a list of instance methods to be used instead of a class method that was registered with the class. Instances with and without methods can be mixed in a class, but they must be added to `dcAddInstance` using a separate class. All instances in the same call must either use methods or not.

The data offset fields are set to zero and the data buffers are not used for this command.

The return address **bufferAddress* must be NULL. The caller is then responsible for subsequently freeing the allocated memory using `dcifree`.

The `DCI_SPI_NAME` method is supported for OS/2 Warp (PowerPC Edition), and is dependent upon the provider supplying a module name to load and a function name to call, along with any appropriate flags in the return value.

the `DCI_SPI_RPC` method is also supported for OS/2 Warp (PowerPC Edition), and relies upon Performance Services to establish a communication link between the supplied method and the consumer's address space.

Upon return `dcAddInstance` will fill the buffer with an array of return structures for each requested instance. The instance identifier for volatile instances is set to 'X'FFFFFFF'.

The summary status of all individual `DCIRetval` structure status members is stored in the `DCIReturn` structure status member. This summary status represents the highest severity of status returned among all `DCIRetval` structures.

The summary status may be one of the following values:

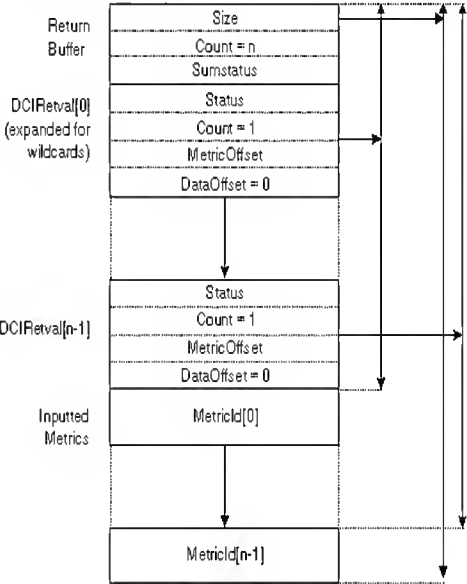
DCI_SUCCESS	All status returned was successful.
DCI_FAILURE	There was at least one failure status.

For each `DCIRetval` structure returned, the status member may contain oen of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
DCI_NOWILDCARD	A wildcard cannot be used in this context.

- DCI_INSTANCEEXISTS
- An instance of this level already exists in the name space.
- DCI_INSTANCENOTPERSISTENT
- If the parent class is not persisten, and the new DCIInstrAttr specifies persistence.
- DCI_INVALIDFIELD
- The associated DCIMethod attributes structure was malformed.
- DCI_METHODOPNOTSUPPORTED
- The requested operation may not be specified in conjunction with this method type.
- DCI_INVALIDMETHODOP
- The operation specified is not a valid operation.
- DCI_INVALIDFIELD
- The associated DCIMethod attributes structure was malformed.
- DCI_NOEXT
- No label has been specified with DCIInstanceAttr field of the DCIMethod.

The following figure shows the return structures for dciAddInstance.



dciAddInstance - Topics

Select an item:

- Syntax
- Parameters
- Returns
- Remarks
- Glossary

dciRemoveInstance

dciRemoveInstance - Syntax

Removes an instance or list of instances from a metric.

```
#include <dcI.h>

DCIMetricId    *metricIdList;
uint32         numIds;
DCIReturn      **bufferAddress;
uint32         bufferSize;
DCIStatus      ulrc;          /* Return code. */

ulrc = dciRemoveInstance(metricIdList, numIds,
                        bufferAddress, bufferSize);
```

dcIRemoveInstance Parameter - metricIdList

metricIdList ([DCIMetricId](#) *) - input
Address of a list of metric class and instance identifiers.

dcIRemoveInstance Parameter - numIds

numIds (uint32) - input
The number of input metric class identifiers.

dcIRemoveInstance Parameter - bufferAddress

bufferAddress ([DCIReturn](#) **) - output
Points to the address of a return value buffer.

dcIRemoveInstance Parameter - bufferSize

bufferSize (uint32) - output
Reserved, set to zero.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using `dciFree`.

dcIRemoveInstance Return Value - ulrc

ulrc ([DCIStatus](#)) - returns
Return code.

dcRemoveInstance returns one of the following values:

DCI_SUCCESS

The DCIReturn structure has been written to the output buffer.

DCI_SYSERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.

DCI_NOWILDCARD

A wildcard cannot be used in this context.

dcRemoveInstance - Parameters

metricIdList ([DCIMetricId](#) *) - input

Address of a list of metric class and instance identifiers.

numIds (uint32) - input

The number of input metric class identifiers.

bufferAddress ([DCIReturn](#) **) - output

Points to the address of a return value buffer.

bufferSize (uint32) - output

Reserved, set to zero.

Must be set to null so that the storage is obtained by the Performance Server. The caller is responsible for freeing the allocated memory using dcFree.

ulrc ([DCIStatus](#)) - returns
Return code.

dcRemoveInstance returns one of the following values:

DCI_SUCCESS

The DCIReturn structure has been written to the output buffer.

DCI_SYSERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.

DCI_NOWILDCARD

A wildcard cannot be used in this context.

dcRemoveInstance - Remarks

dcRemoveInstance removes a list of instances from the metrics name space. This routine takes as input a buffer with an array of DCIMetricId structures. dcRemoveInstance determines if each instanceid in the metricIdList is currently registered in the metrics name space, expands any instance wildcards in the metricOffset buffer, determines if each expanded instance is currently registered in the metrics name space, and unregisters it from the name space. For each expanded metric instance a DCIRetval references the expanded DCIMetricId structure and the dataOffset member is set to zero and the data buffers are not used. If so configured, this will lead to a final data notification to all interested DCI consumers that have the instance open.

The return address *bufferAddress must be NULL. The caller is then responsible for subsequently freeing the allocated memory using dcFree.

The datumId field of the DCIMetricId is ignored.

The instanceId for volatile instances is the wildcard value.

Upon return, dcRemoveInstance will produce a DCIReturn structure at the address pointed to by bufferAddress. Each DCIRetval structure

within the DCIReturn structure reflects the status of each instance removed. The metricOffset member of each DCIRetval specifies a location relative to bufferaddress and references the DCIMetricId structure.

The summary status of all individual DCIRetval structure status members is stored in the DCIReturn structure status member. This summary status represents the highest severity of status returned among all DCIRetval structures.

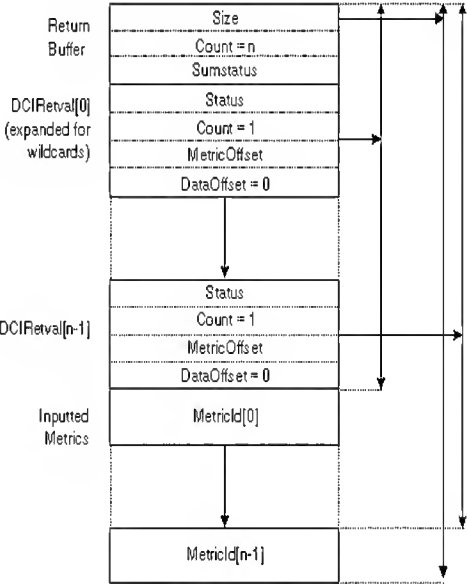
The summary status may contain one of the following values:

DCI_SUCCESS	All status returned was successful.
DCI_FAILURE	There was at least one failure status.

For each DCIRetval structure returned, the status member may contain one of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
DCI_NOINSTANCE	The requested instance identifier is not in the name space.
DCI_NOCLASS	The requested metric class identifier is not present in the name space.

The following figure shows the return structures for dciRemoveInstance.



dciRemoveInstance - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

DCI_SPI_NAMEfunctionname

DCI_SPI_NAMEfunctionname - Syntax

The DCI_SPI_NAME method is used when providers supply a shared library as an interface to obtain and prepare information.

```
#include <dc_i.h>

DCIMetricId    mid;           /* List of metric ids. */
DCIMetricId    numofmetids;   /* Number of metric ids in list. */
uint32         calltype;     /* The function call type. */
DCIReturn      **retbuffer;
uint32         retbufsize;
void           **databuffer;
uint32         *databufsize;
DCIStatus      Return Value;

Return Value = DCI_SPI_NAMEfunctionname(
    mid, numofmetids, calltype,
    retbuffer, retbufsize, databuffer,
    databufsize);
```

DCI_SPI_NAMEfunctionname Parameter - mid

mid ([DCIMetricId](#)) - input
List of metric ids.

DCI_SPI_NAMEfunctionname Parameter - numofmetids

numofmetids ([DCIMetricId](#)) - input
Number of metric ids in list.

DCI_SPI_NAMEfunctionname Parameter - calltype

calltype (uint32) - input
The function call type.

May be one of the following types:

DCI_OPEN

DCI_VOLATILE

This is used to indicate that instances are too transient to be placed in the name space and a provider function is given to resolve instance information.

DCI_GETDATA

DCI_CLOSE

DCI_GETINSTATTR

DCI_LISTINSTANCEID

DCI_SPI_NAMEfunctionname Parameter - retbuffer

retbuffer ([DCIReturn **](#)) - output
A pointer to the address of a return value buffer.

DCI_SPI_NAMEfunctionname Parameter - retbufsize

retbufsize (uint32) - input
Reserved, set to zero.

DCI_SPI_NAMEfunctionname Parameter - databuffer

databuffer - output
The return buffer.

DCI_SPI_NAMEfunctionname Parameter - databufsize

databufsize (uint32 *) - output
A pointer to the size of the data buffer.

DCI_SPI_NAMEfunctionname Return Value - Return Value

Return Value ([DCIStatus](#)) - returns
`dciListClassId` returns one of the following values:

- `DCI_SUCCESS`
The DCIReturn structure has been written to the output buffer.
 - `DCI_NOTINITIALIZED`
The requester has not yet initialized the DCI.
 - `DCI_SYSERROR`
A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
 - `DCI_INVALIDARG`
One of the input arguments is invalid. For example, a negative value was used for `numIds` or `bufferSize`. Also, either `metricIdList` could not be read or buffer could not be written.
 - `DCI_BADHANDLE`
The handle provided is not currently open.
-

DCI_SPI_NAMEfunctionname - Parameters

mid ([DCIMetricId](#)) - input
List of metric ids.

numofmetids ([DCIMetricId](#)) - input
Number of metric ids in list.

calltype (uint32) - input
The function call type.

May be one of the following types:

DCI_OPEN

DCI_VOLATILE

This is used to indicate that instances are too transient to be placed in the name space and a provider function is given to resolve instance information.

DCI_GETDATA

DCI_CLOSE

DCI_GETINSTATTR

DCI_LISTINSTANCEID

retbuffer ([DCIReturn **](#)) - output
A pointer to the address of a return value buffer.

retbufsize (uint32) - input
Reserved, set to zero.

databuffer - output
The return buffer.

databufsize (uint32 *) - output
A pointer to the size of the data buffer.

Return Value ([DCIStatus](#)) - returns
dciListClassId returns one of the following values:

DCI_SUCCESS

The DCIReturn structure has been written to the output buffer.

DCI_NOTINITIALIZED

The requester has not yet initialized the DCI.

DCI_SYSEERROR

A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.

DCI_INVALIDARG

One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.

DCI_BADHANDLE

The handle provided is not currently open.

DCI_SPI_NAMEfunctionname - Remarks

The method field within the [DCIMethod](#) structure for the DCI_SCP_NAME method consists of a null-terminated ASCII string which contains the library name, followed by a null-terminated ASCII string which contains the function name. Providers must specify the functions they support in the calltype field. For example, DCI_OPEN, DCI_GETDATA, DCI_LISTINSTANCEID, DCI_GETINSTATTR, and DCI_CLOSE might be selected by a provider who had setup and cleanup to do prior to being called for data. DCI_VOLATILE would designate a method which returned valid metric ids for volatile instances and their instance attributes (because their identifiers are too volatile to list separately in the name space). Tasks and threads need to support DCI_VOLATILE due to their dynamic nature.

The summary status of all individual [DCIRetVal](#) structure status members is stored in the [DCIReturn](#) structure status member. This summary status represents the highest severity of status returned among all [DCIRetVal](#) structures.

The summary status may be one of the following values:

DCI_SUCCESS

All status returned was successful.

DCI_FAILURE

There was at least one failure status.

For each [DCIRetval](#) structure returned, the status member may contain one of the following values:

DCI_SUCCESS	The request succeeded and there may be associated data.
DCI_NOCLASS	The class identifier is not present in the name space.
DCI_NOINSTANCE	The requested instance identifier is not in the name space.
DCI_CLASSES_CHANGED	This new class has been added within the scope of a wildcarded class list.
DCI_INSTANCES_CHANGED	This new Instance has been added within the scope of a wildcarded class request.

DCI_SPI_NAMEfunctionname - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DCI_SPI_RPCfunctionname

DCI_SPI_RPCfunctionname - Syntax

The DCI_SPI_RPC method is used by providers who want Performance Services to establish a communication path between the consumer's address space and the provider's address space in order to call a provider-supplied function within the provider's address space to obtain the data information for the metric id registered.

```
#include <dc_i.h>

char      *metricidstr;    /* A metricidstr whose delimiter is first. */
void      *retbuffer;     /* Return buffer. */
uint32    *userword;
uint32    calltype;       /* The function call type. */
uint32    datasize;
DCIStatus  Return Value;

Return Value = DCI_SPI_RPCfunctionname(
    metricidstr, retbuffer, userword,
    calltype, datasize);
```

DCI_SPI_RPCfunctionname Parameter - metricidstr

metricidstr (char *) - input
A metricidstr whose delimiter is first.

DCI_SPI_RPCfunctionname Parameter - retbuffer

retbuffer - output
Return buffer.

DCI_SPI_RPCfunctionname Parameter - userword

userword (uint32 *) - input
The userword supplied by the provider at registration time.

DCI_SPI_RPCfunctionname Parameter - calltype

calltype (uint32) - input
The function call type.

May be one of the following types:

- DCI_GETDATA
-

DCI_SPI_RPCfunctionname Parameter - datasize

datasize (uint32) - output
The return buffer size.

DCI_SPI_RPCfunctionname Return Value - Return Value

Return Value ([DCIStatus](#)) - returns
dciGetData returns one to the following values:

- | | |
|--------------------|---|
| DCI_SUCCESS | The DCIReturn structure has been written to the output buffer. |
| DCI_NOTINITIALIZED | The requester has not yet initialized the DCI. |
| DCI_SYSERROR | A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application. |
| DCI_INVALIDARG | One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written. |
| DCI_BADHANDLE | The handle provided is not currently open. |

DCI_SPI_RPCfunctionname - Parameters

metricidstr (char *) - input
A metricidstr whose delimiter is first.

retbuffer - output
Return buffer.

userword (uint32 *) - input
The userword supplied by the provider at registration time.

calltype (uint32) - input
The function call type.

May be one of the following types:

- DCI_GETDATA

datasize (uint32) - output
The return buffer size.

Return Value ([DCIStatus](#)) - returns
dciGetData returns one of the following values:

DCI_SUCCESS	The DCIReturn structure has been written to the output buffer.
DCI_NOTINITIALIZED	The requester has not yet initialized the DCI.
DCI_SYSERROR	A internal error has occurred (such as a shortage of resources) that may be beyond the control of the application.
DCI_INVALIDARG	One of the input arguments is invalid. For example, a negative value was used for numIds or bufferSize. Also, either metricIdList could not be read or buffer could not be written.
DCI_BADHANDLE	The handle provided is not currently open.

DCI_SPI_RPCfunctionname - Remarks

The provider's function accepts as input the metricid, userword, calltype and datasize it passed at registration time, as well as the return buffer obtained for its use by Performance Services. The DCI_SPI_RPC method is the only method supported by the dciEZRegister API. The DCI_SPI_RPC method only supports the calltype of DCI_GETDATA. Performance Services keeps a table with the metricid, metricidstr, function address, userword, calltype, and datasize passed at registration time which is checked by Performance Services during a data request.

DCI_SPI_RPCfunctionname - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DCI Data Types

This section describes the following data types that are used with the DCI functions:

- [DCIClassId](#)
- [DCIClassAttr](#)
- [DCIDataAttr](#)
- [DCIInstAttr](#)
- [DCIInstanceId](#)
- [DCIInstanceType](#)
- [DCIInstLevel](#)
- [DCILabel](#)
- [DCILocalExt](#)
- [DCIMethod](#)
- [DCIMetricId](#)
- [DCIRetval](#)
- [DCIReturn](#)
- [DCIStatus](#)
- [DCI_SPINameData](#)
- [DCI_SPIRPCData](#)
- [DCIUnit](#)
- [UMAArrayDescr](#)
- [UMADataType](#)
- [UMAElemDescr](#)
- [UMATextDescr](#)
- [UMATextString](#)
- [UMAVarArrayDescr](#)
- [UMAVarLenData](#)
- [UMAVarLenDescr](#)

DCIClassId

Data structure for a class identifier.

```
typedef struct _DCIClassId {
    uint32      size;           /* Size of the fixed and variable length portions. */
    UMAArrayDescr identifier;    /* Descriptor for the variable-length identifier. */
    UMAVarLenData data;         /* Start of the data section. */
} DCIClassId;

typedef DCIClassId *DCIClassId;
```

Class names are a sequence of words with one word per metric class level.

DCIClassId Field - size

size (uint32)
Size of the fixed and variable length portions.

DCIClassId Field - identifier

identifier ([UMAArryDescr](#))
Descriptor for the variable-length identifier.

DCIClassId Field - data

data ([UMAVarLenData](#))
Start of the data section.

DCIClassAttr

Data structure that describes the attributes of a class.

```
typedef struct _DCIClassAttr {
    uint32      size;          /* Total structure size. */
    uint32      flags;         /* Special class state. */
    UMAVarLenDescr reserved;   /* Reserved, set to 0 for Release 1. */
    UMAVarLenDescr method;
    UMAVarLenDescr label;
    UMAArryDescr    instlevel;
    UMAVarArrayDescr dataAttr;
    UMAVarArrayDescr reserved; /* Reserved, set to 0 for Release 1. */
    UMAElementDescr reserved; /* Reserved, set to 0 for Release 1. */
    UMAVarLenData    data;     /* Start of data section. */
} DCIClassAttr;

typedef DCIClassAttr *DCIClassAttr;
```

The DCIClassAttr attributes structure describes the attributes of a class. These attributes are set by the provider of a class when registering the class with [dciRegister](#) or [dciEZRegister](#), and cannot be changed unless the class is unregistered with [dciUnregister](#) or [dciEZUnregister](#). They may be retrieved by any consumer or provider using [dciGetClassAttributes](#).

The DCIClassAttr structure provides a label for the class, every individual datum supported by the class, and the structure of the class' instance space. Several of the fields in the structure have a variable size.

Each class level registered has a class attribute structure within the name space along with a structure to define its level name.

DCIClassAttr Field - size

size (uint32)
Total structure size.

DCIClassAttr Field - flags

flags (uint32)
Special class state.

This field can contain a sequence of bit-mapped flags. The values of these flags are:

DCI_ENABLED

	This class is enabled and available for use. All classes registered are considered enabled. This bit must be set for all registrations of all metrics.
DCI_DISABLED	
	This class is disabled. This status is not supported. All classes registered are considered enabled.
DCI_NOTIMPLEMENTED	
	A non-implemented class.
DCI_NOTAPPLICABLE	
	This class is not applicable to the measured system.
DCI_OBSOLETE	
	This class is no longer supported on the measured system.
DCI_PROVIDER_INSTANCE	
	The provider does not register instances with the DCI. All instance requests go directly to the provider's method. This is used for volatile instances. Volatile instances are used for classes where registering each instance is regarded as too time consuming considering the life of the instance. For example, threads, task and memory objects. Change notification of volatile instances is not supported.

DCIClassAttr Field - reserved

reserved ([UMAVarLenDescr](#))
Reserved, set to 0 for Release 1.

DCIClassAttr Field - method

method ([UMAVarLenDescr](#))
Descriptor for [DCIMethod](#) structure.

DCIClassAttr Field - label

label ([UMAVarLenDescr](#))
Descriptor for [DCILabel](#) structure.

DCIClassAttr Field - instlevel

instlevel ([UMAArrayDescr](#))
Descriptor for a [DCIInstLevel](#) array.

DCIClassAttr Field - dataAttr

dataAttr ([UMAVarArrayDescr](#))
Descriptor for a [DCIDataAttr](#) array.

DCIClassAttr Field - reserved

reserved ([UMAVarArrayDescr](#))
Reserved, set to 0 for Release 1.

DCIClassAttr Field - reserved

reserved ([UMAElementDescr](#))
Reserved, set to 0 for Release 1.

DCIClassAttr Field - data

data ([UMAVarLenData](#))
Start of data section.

DCIDataAttr

Data structure for an individual datum within a class.

```
typedef struct _DCIDataAttr {
    uint32      size;      /* Total structure size. */
    uint32      datumId;   /* The individual metric identifier. */
    uint32      type;      /* Data type for this metric. */
    DCIUnit     units;     /* Units for this metric. */
    uint32      flags;     /* Metric characteristics. */
    uint32      offset;    /* Byte offset from the beginning of the returned class data. */
    UMAVarLenDescr label;
    UMAVarLenData data;    /* Start of the data area. */
} DCIDataAttr;

typedef DCIDataAttr *DCIDataAttr;
```

The `DCIDataAttr` structure is used to describe an individual datum within a class. All of the data attribute structures are kept at the class type node along with the rest of the class attribute structures.

DCIDataAttr Field - size

size (uint32)
Total structure size.

DCIDataAttr Field - datumId

datumId (uint32)
The individual metric identifier.

DCIDataAttr Field - type

type (uint32)
Data type for this metric.

Possible values are:

- uint32
- uint64

DCIDataAttr Field - units

units (DCIUnit)
Units for this metric.

This field is used to indicate what the metric is measuring. The purpose of this field is to tell the measurement application (the DCI consumer) what type of size, time, or count is being measured. Below is a list of valid units.

The following constants are used with the time, count, size and information units:

DCU_UNITS_TIME	0x100000
DCU_UNITS_COUNT	0x200000
DCU_UNITS_SIZE	0x300000
DCU_UNITS_INFO	0x400000

Possible values for time units are:

DCI_SECS	0x01	DCI_UNITS_TIME
DCI_MILLISECS	0x02	DCI_UNITS_TIME
DCI_MICROSECS	0x03	DCI_UNITS_TIME
DCI_NANOSECS	0x04	DCI_UNITS_TIME
DCI_PICOSECS	0x05	DCI_UNITS_TIME
DCI_TICKS	0x06	DCI_UNITS_TIME

Possible values for count units are:

DCI_COUNT	0x01	DCI_UNITS_COUNT
DCI_EVENT	0x02	DCI_UNITS_COUNT
DCI_PAGES	0x03	DCI_UNITS_COUNT
DCI_BLOCKS	0x04	DCI_UNITS_COUNT
DCI_CHARACTERS	0x05	DCI_UNITS_COUNT
DCI_QLENGTH	0x06	DCI_UNITS_COUNT
DCI_PROCESSES	0x07	DCI_UNITS_COUNT
DCI_TASKS	0x08	DCI_UNITS_COUNT
DCI_THREADS	0x09	DCI_UNITS_COUNT
DCI_JOBS	0x0A	DCI_UNITS_COUNT
DCI_USERS	0x0B	DCI_UNITS_COUNT
DCI_TRANSACTIONS	0x0C	DCI_UNITS_COUNT
DCI_MESSAGES	0x0D	DCI_UNITS_COUNT

DCI_SESSIONS	0x0E DCI_UNITS_COUNT
DCI_STREAMSMODULES	0x0F DCI_UNITS_COUNT
DCI_STREAMSHEADS	0x10 DCI_UNITS_COUNT
DCI_STREAMSMSGS	0x11 DCI_UNITS_COUNT
DCI_PACKETS	0x12 DCI_UNITS_COUNT
DCI_INODES	0x13 DCI_UNITS_COUNT
DCI_FILES	0x14 DCI_UNITS_COUNT
DCI_FILESYSTEMS	0x15 DCI_UNITS_COUNT
DCI_READS	0x16 DCI_UNITS_COUNT
DCI_WRITES	0x17 DCI_UNITS_COUNT
DCI_SEEKS	0x18 DCI_UNITS_COUNT
DCI_IOCTLs	0x19 DCI_UNITS_COUNT
DCI_CONNECTIONS	0x1A DCI_UNITS_COUNT
DCI_RETRIES	0x1B DCI_UNITS_COUNT
DCI_MOUNTS	0x1C DCI_UNITS_COUNT
DCI_REWINDS	0x1D DCI_UNITS_COUNT
DCI_POSITIONINGS	0x1E DCI_UNITS_COUNT
DCI_MARKS	0x1F DCI_UNITS_COUNT
DCI_PORTS	0x20 DCI_UNITS_COUNT
DCI_PROCESSORS	0x21 DCI_UNITS_COUNT
DCI_DISKS	0x22 DCI_UNITS_COUNT
DCI_NETS	0x23 DCI_UNITS_COUNT
DCI_SLINES	0x24 DCI_UNITS_COUNT
DCI_BUSSES	0x25 DCI_UNITS_COUNT
DCI_CHANNELS	0x26 DCI_UNITS_COUNT
DCI_NOUNITS	0x27 DCI_UNITS_COUNT
Possible values for size units are:	
DCI_BYTES	0x01 DCI_UNITS_SIZE
DCI_KBYTES	0x02 DCI_UNITS_SIZE
DCI_MBYTES	0x03 DCI_UNITS_SIZE
DCI_GBYTES	0x04 DCI_UNITS_SIZE
DCI_TBYTES	0x05 DCI_UNITS_SIZE
Possible values for information units are:	
DCI_CPU	0x01 DCI_UNITS_INFO
DCI_MEMORY	0x02 DCI_UNITS_INFO
DCI_PRIORITY	0x03 DCI_UNITS_INFO
DCI_THREAD	0x04 DCI_UNITS_INFO
DCI_TASK	0x05 DCI_UNITS_INFO
DCI_DATA	0x06 DCI_UNITS_INFO
DCI_BOOLEAN	0x07 DCI_UNITS_INFO
DCI_ADDRESS	0x08 DCI_UNITS_INFO
DCI_COUNT	0x09 DCI_UNITS_INFO
DCI_PROTECT	0x0A DCI_UNITS_INFO
DCI_OBJECT_NAME	0x0B DCI_UNITS_INFO
DCI_OFFSET	0x0C DCI_UNITS_INFO
DCI_BYTES	0x0D DCI_UNITS_INFO

DCIDataAttr Field - flags

flags (uint32)
Metric characteristics.

Describes other characteristics of the metric in terms of what method operations the provider will support on the metric.

The possible values are:

- DCI_QUERYABLE

DCIDataAttr Field - offset

offset (uint32)

Byte offset from the beginning of the returned class data.

DCIDataAttr Field - label

label ([UMAVarLenDescr](#))
Description for the variable-length [DCILabel](#)

DCIDataAttr Field - data

data ([UMAVarLenData](#))
Start of the data area.

DCIInstAttr

Data structure for an instance attribute.

```
typedef struct _DCIInstAttr {  
    uint32      size;          /* Total structure size. */  
    uint32      flags;         /* Instance flags. */  
    UMAVarLenDescr reserved;    /* Reserved, set to 0 for Release 1. */  
    UMAElemDescr  localExt;    /* Local extensions used by implementation-specific information. */  
    UMAVarLenDescr label;  
    UMAVarLenData data;  
} DCIInstAttr;  
  
typedef DCIInstAttr *DCIInstAttr;
```

The DCIInstAttr structure can be used to obtain a particular instance's label.

DCIInstAttr Field - size

size (uint32)
Total structure size.

DCIInstAttr Field - flags

flags (uint32)
Instance flags.

The possible values are:

DCI_PERSISTENT_INSTANCE

The instance should not be removed from the name space (i.e. unregistered) when the process that registered this instance terminates.

DCIInstAttr Field - reserved

reserved ([UMAVarLenDescr](#))

Reserved, set to 0 for Release 1.

DCIInstAttr Field - localExt

localExt ([UMAElementDescr](#))

Local extensions used by implementation-specific information.

localExt is a descriptor to [DCILocalExt](#).

DCIInstAttr Field - label

label ([UMAVarLenDescr](#))

Description for the variable-length [DCILabel](#)

DCIInstAttr Field - data

data ([UMAVarLenData](#))

Start of the data area for [DCILabel](#) and [DCILocalExt](#).

DCIInstancedId

Data structure for an instance identifier.

```
typedef struct _DCIInstanceId {
    uint32      size;           /* Size of the instance id in bytes. */
    uint32      levelWild;     /* Wildcard bitmap, 32 levels, instance id. */
    UMAArryDescr instances;     /* Descriptor for the variable-length instance array. */
    UMAVarLenData data;        /* The variable-length instance array. */
} DCIInstanceId;

typedef DCIInstanceId *DCIInstanceId;
```

An instance identifier is used to select one of a set of otherwise identical groups of metrics. For example, if one requests ClassId 254.1.3.1 with wildcarded instance ids, all of the thread information for all of the tasks currently active on the system is returned. The return structure's InstancedId array contains the task and thread values which identify a particular data area in the data buffer. Multiple instances can be

represented within a single metric id.

Wildcarding is done via a bit mask in the `DCIInstanceld` structure. For example, instance id level one uses the mask 1, instance id level two uses the mask 2, and instance id level 3 uses the mask 4. Masks for multiple levels can be OR'd together.

DCIInstanceld Field - size

size (uint32)

Size of the instance id in bytes.

DCIInstanceld Field - levelWild

levelWild (uint32)

Wildcard bitmap, 32 levels, instance id.

See the Wildcarding figure under [Metrics Name Space](#).

DCIInstanceld Field - instances

instances ([UMAArrayDescr](#))

Descriptor for the variable-length instance array.

DCIInstanceld Field - data

data ([UMAVarLenData](#))

The variable-length instance array.

DCIInstanceType

DCI instance types.

Possible values are:

1	DCI_SINGLEINST	A single instance of value "0" exists
2	DCI_WORKINFO	UMA_WORKINFO enumeration
3	DCI_WORKID	Data associated with DCI_WORKINFO
4	DCI_MSG_QUEUE	

5	DCI_SEMAPHORE	
6	DCI_SHR_SEGMENT	
7	DCI_PROCESSOR	Processor number
8	DCI_FSGROUP	
9	DCI_MOUNTPOINT	
10	DCI_INODE	Inode number
11	DCI_DISKID	Disk device number
12	DCI_BUCKET_NO	
13	DCI_DISKPARTITION	
14	DCI_ACCESS_PORT	
15	DCI_DEVICE	Generic device number
16	DCI_KERNEL_TABLES	
17	DCI_CHANNEL	Channel number
18	DCI_IOP	IO processor number
19	DCI_PATH	
20	DCI_SYSCALL	System call number
21	DCI_ENUMERATION	
22	DCI_STREAMS	
23	DCI_CONTROLLERID	Controller number
24	DCI_SCHED_CLASS	Scheduling class type
25	DCI_LOGICALVOL	
26	DCI_REMOTE_FSTYPES	
27	DCI_IPADDR	
28	DCI_FILESERVER_COMMAND	
29	DCI_FILECLIENT_COMMAND	
30	DCI_SERVER_COMMAND	
31	DCI_CLIENT_COMMAND	
32	DCI_MEMOBJECT_ID	

```
typedef uint32 DCIInstanceType;
```

DCIInstLevel

Data structure for an instance level.

```
typedef struct _DCIInstLevel {
    UMADataType    type;    /* Type of the instance level value. */
    DCIInstanceType itype;  /* The instance type. */
    uint32         size;    /* Size of the instance level value in bytes. */
} DCIInstLevel;

typedef DCIInstLevel *DCIInstLevel;
```

The DCIInstLevel structure is used in an instantiated DCIClassAttr structure to describe each instance level. This structure allows classes to have multiple, self-described instance levels. For example, multi-level instances can be used to categorize metrics as "per-processor, per-disk I/O metrics" in a multiprocessor system with asymmetric I/O where disk drives are partitioned between processors.

Each level within an instance can have a different size.

DCIInstLevel Field - type

type (UMADDataType)

Type of the instance level value.

DCIInstLevel Field - itype

itype (DCIInstanceType)

The instance type.

The instance type one uses when working with a particular instance value.

DCIInstLevel Field - size

size (uint32)

Size of the instance level value in bytes.

The number of bytes used for an instance level whose type is described in this structure. This size must be a multiple of 4. (This size should not be confused with the size of the DCIInstLevel structure itself.)

DCILabel

The label attribute structure contains one form of the label.

```
typedef struct _DCILabel {
    uint32          size;          /* The total structure size. */
    UMATextString   ascii;
    UMAElementDesc reserved; /* Reserved. Set to 0 for Release 1. */
    UMAMVarLenData  data;        /* Start of the data. */
} DCILabel;
```

```
typedef DCILabel *DCILabel;
```

Labels can be used to assign metric names that are more meaningful than the metric identifier. Applications may then search the name space for a metric using the metric's label, rather than the metric id.

The label is a variable length null-terminated ASCII string, padded on the right for a word boundary if necessary. This is to ensure that the *size* field is always a multiple of 4 bytes.

Labels should be made unique using following criteria:

- Metric class labels should be unique within each level of the DCI name space hierarchy.

- Instance labels should be unique within each level of the DCI name space hierarchy.
- Individual metric (datum) labels should be unique with the class.

Maintaining unique labels is the responsibility of the entity that registers the class or adds a new instance. The Performance Services function will not return an error status if the label is not unique.

The provider must register a label with each name space entity (metric class, metric instance or metric datum) before an application may use the label to search the name space.

Once a metric class, instance or datum has been labeled, the metric provider cannot relabel it unless the metric has been completely unregistered.

DCILabel Field - size

size (uint32)
The total structure size.

DCILabel Field - ascii

ascii ([UMATextString](#))
Descriptor for the variable-length [UMATextString](#) for ASCII.

DCILabel Field - reserved

reserved ([UMAElementDescr](#))
Reserved. Set to 0 for Release 1.

DCILabel Field - data

data ([UMAVarLenData](#))
Start of the data.

DCILocalExt

Data structure used to identify process information of an instance.

```
typedef struct _DCILocalExt {
    uint32      tid;      /* Thread id. */
    uint32      pid;      /* Process id. */
    UMAVarLenDescr  pathname;
    UMAVarLenData  data;    /* Start of the data area. */
}
```

```
} DCILocalExt;  
  
typedef DCILocalExt *DCILocalExt;
```

The local extensions are used to further specify the instance in the [DCIInstAttr](#) structure. This additional information can help identify a particular process' information. For non-volatile instances which are already unique to the microkernel, the term SYSTEM is used for the path name, and tid and pid are set to 0. In the case of the default pager, the load path of the default pager is given since there can be multiple pagers on the system. In the case of the OS/2 loader, the load path of the OS/2 loader is given. In the case of volatile instances, the process id and load path is given for process information, and the task id, process id and load path is given for task information.

DCILocalExt Field - tid

tid (uint32)

Thread id.

Set to 0 for non-volatile instances.

DCILocalExt Field - pid

pid (uint32)

Process id.

Set to 0 for non-volatile instances.

DCILocalExt Field - pathname

pathname ([UMAVarLenDescr](#))

Descriptor for the variable-length [UMATextString](#) for ASCII.

Set to SYSTEM for non-volatile instances.

For the default pager, set to the load path of the default pager.

For the OS/2 loader, set to the load path of the OS/2 loader.

DCILocalExt Field - data

data ([UMAVarLenData](#))

Start of the data area.

DCIMethod

Data structure for a Performance Services method.

```
typedef struct _DCIMethod {
    uint32      size;      /* Method structure size in bytes. */
    uint32      type;      /* Type of method. */
    UMAElementDescr method; /* The variable-length descriptor for type. */
    UMALenData   data;     /* Data for the method. */
} DCIMethod;

typedef DCIMethod *DCIMethod;
```

The process by which Performance Services retrieves metric information and the provider specifies metric information is called a method. The provider must define and provide the method at registration. The DCIMethod structure is used to define the method. The method can be registered in either the lowest class level in a class with [dciRegister](#), or the lowest instance level with [dciAddInstance](#) or [dciEZRegister](#).

If a method is defined in both the class and instance of a metricid, the instance method takes precedence. A metric itself is never stored in the name space; only the method to retrieve it is stored.

DCI_SPI_NAME Methods

The DCI_SPI_NAME method is used when providers supply a shared library as an interface to obtain and prepare information. The method field within the DCIMethod structure for the DCI_SCP_NAME method consists of a null-terminated ASCII string which contains the library name, followed by a null-terminated ASCII string which contains the function name. Providers must specify the functions they support in the type field. For example, DCI_OPEN, DCI_GETDATA, DCI_LISTINSTANCEID, DCI_GETINSTATTR, and DCI_CLOSE might be selected by a provider who had setup and cleanup to do prior to being called for data. DCI_VOLATILE would designate a method which returned valid metric ids for volatile instances and their instance attributes (because their identifiers are too volatile to list separately in the name space). Tasks and threads need to support DCI_VOLATILE due to their dynamic nature.

See [DCI_SPI_NAMEfunctionname](#) for the invocation parameters of a DCI_SPI_NAME function .

DCI_SPI_RPC Methods

The DCI_SPI_RPC method is used by providers who want Performance Services to establish a communication path between the consumer's address space and the provider's address space in order to call a provider-supplied function within the provider's address space to obtain the data information for the metric id registered. The provider's function accepts as input the metricid, userword, calltype and datasize it passed at registration time, as well as the return buffer obtained for its use by Performance Services. The DCI_SPI_RPC method is the only method supported by the dciEZRegister API. The DCI_SPI_RPC method only supports the calltype of DCI_GETDATA. Performance Services keeps a table with the metricid, metricidstr, function address, userword, calltype, and datasize passed at registration time which is checked by Performance Services during a data request.

See [DCI_SPI_NAMEfunctionname](#) for the invocation parameters of a DCI_SPI_RPC function .

DCIMethod Field - size

size (uint32)
Method structure size in bytes.

DCIMethod Field - type

type (uint32)
Type of method.

The following method types are available:

DCI_SPI_NAME
Method type for [DCI_SPINameData](#)

DCI_SPI_RPC
Method type for [DCI_SPIRPCData](#)

DCIMethod Field - method

method (UMAElemDescr)

The variable-length descriptor for type.

DCI_SPINameData's offset and size or DCI_SPIRPCData's offset and size.

DCIMethod Field - data

data (UMAVallLenData)

Data for the method.

Dependent upon method type.

DCIMetricId

Data structure for a named metric identifier.

```
typedef struct _DCIMetricId {
    uint32          size;           /* Size of whole structure for the metricId. */
    UMAVarLenDescr  classId;       /* Descriptor for the variable length DCIClassId. */
    UMAVarLenDescr  instanceId;    /* Descriptor for the variable length DCIInstanceId. */
    DCIDatumId      datumId;       /* Reserved. */
    UMAVarLenData    data;         /* The data of DCIClassId and DCIInstanceId. */
} DCIMetricId;

typedef DCIMetricId *DCIMetricId;
```

A completed named metric consists of the DCIClassId, the DCIInstanceId, and a DCIDatumId value.

DCIMetricId Field - size

size (uint32)

Size of whole structure for the metricId.

DCIMetricId Field - classId

classId (UMAVarLenDescr)

Descriptor for the variable length DCIClassId.

DCIMetricId Field - instanceld

instanceld (UMAVarLenDescr)

Descriptor for the variable length DCIInstanceld.

DCIMetricId Field - datumId

datumId (DCIDatumId)

Reserved.

DCIMetricId Field - data

data (UMAVarLenData)

The data of DCIClassId and DCIInstanceld.

DCIRetval

A structure that contains return information for an individual metric or class identifier.

```
typedef struct _DCIRetval {
    uint32    status;        /* The status for this metric. */
    uint32    count;         /* The number of metric ids. */
    uint32    metricOffset;  /* Offset from the return buffer address. */
    uint32    dataOffset;    /* Offset from the data buffer address. */
} DCIRetval;
```

```
typedef DCIRetval *DCIRetval;
```

The DCIReturn structure contains an array of DCIRetval structures.

DCIRetval Field - status

status (uint32)

The status for this metric.

The status values are defined using the following constants:

DCI_FAILURE	0x4002A000
DCI_INFORMATIONAL	0x2002A000
DCI_FATAL	0x0802A000

status may be one of the following values:

DCI_SUCCESS	0x1002A000
-------------	------------

```

DCI_CLASSCHANGED
    (DCI_INFORMATIONAL | 0x01)
DCI_CLASSEXISTS
    (DCI_FAILURE | 0x08)
DCI_CLASSNOTEMPTY
    (DCI_FAILURE | 0x18)
DCI_CLASSNOTPERSISTENT
    (DCI_FAILURE | 0x16)
DCI_INSTANCECHANGED
    (DCI_INFORMATIONAL | 0x02)
DCI_INSTANCEEXISTS
    (DCI_FAILURE | 0x09)
DCI_INSTANCENOTPERSISTENT
    (DCI_FAILURE | 0x17)
DCI_INVALIDMETHODOP
    (DCI_FAILURE | 0x14)
DCI_METHODOPNOTSUPPORTED
    (DCI_FAILURE | 0x13)
DCI_METHODTYPEUNAVAILABLE
    (DCI_FAILURE | 0x0b)
DCI_NOCLASS
    (DCI_FAILURE | 0x02)
DCI_NOINSTANCE
    (DCI_FAILURE | 0x03)
DCI_NOMETRIC
    (DCI_FAILURE | 0x04)
DCI_NOTEXT
    (DCI_FAILURE | 0x06)
DCI_NOWILDCARD
    (DCI_FAILURE | 0x07)
DCI_SYSEERROR
    (DCI_FATAL | 0x04)
DCI_INVALIDFIELD
    (DCI_FATAL | 0x06)

```

DCIRetval Field - count

count (uint32)

The number of metric ids.

A count of the number of returned data items for the request. This is set to 1 due to one Retval per metricid or classid expansion.

DCIRetval Field - metricOffset

metricOffset (uint32)

Offset from the return buffer address.

Points to a [DCIMetricId](#) or [DCIClassId](#) structure.

DCIRetval Field - dataOffset

dataOffset (uint32)

Offset from the data buffer address.

DCIReturn

A structure that contains return information from a DCI routine.

```
typedef struct _DCIReturn {
    uint32     size;           /* The total size of the return buffer in bytes. */
    uint32     count;         /* The number of metric ids for which information was requested. */
    uint32     sumstatus;     /* The summary status. */
    DCIRetval  retval;
} DCIReturn;

typedef DCIReturn *DCIReturn;
```

See [Return Status and Structures](#) for more information on DCIReturn.

DCIReturn Field - size

size (uint32)
The total size of the return buffer in bytes.

DCIReturn Field - count

count (uint32)
The number of metric ids for which information was requested.

DCIReturn Field - sumstatus

sumstatus (uint32)
The summary status.

May be one of the following values:

DCI_SUCCESS	(0x1002A000) All status returned was successful.
DCI_FAILURE	(0x4002A000) There was at least one failure status returned in a DCIRetval structure.

DCIReturn Field - retval

retval ([DCIRetval](#))
An array of DCIRetval structures, one for each metric requested.

DCIStatus values are defined using the following constant:

DCIStatus may contain one of the following values:

```
typedef uint32 DCIStatus;
```

DCI_SPINameData Field - calltype

calltype (uint32)
The call type.

DCI_SPINameData Field - data

data (UMAVarLenData)
Start of the data area.

DCI_SPIRPCData

DCISPIRPCData structure referenced by DCIMethod.

```
typedef struct _DCI_SPIRPCData {  
    uint32    calltype; /* The call type. */  
} DCI_SPIRPCData;  
  
typedef DCI_SPIRPCData *DCI_SPIRPCData;
```

DCI_SPIRPCData Field - calltype

calltype (uint32)
The call type.

May be on of the following call types:

- DCI_GETDATA
-

DCIUnit

Used to indicate what type of units a metric is measuring.

```
typedef uint32 DCIUnit;
```

UMAArrayDescr

Descriptor for variable-length array of fixed-sized elements.

```
typedef struct _UMAArraryDescr {
    uint32    offset; /* Offset to beginning of data. */
    uint32    count;  /* Count of elements in the array. */
    uint32    size;   /* Size of each element in the array. */
} UMAArraryDescr;

typedef UMAArraryDescr *UMAArraryDescr;
```

Support for random access to variable length members of structures requires that the address of these members is derived from fixed size structures at known offsets within the main structure definition. There are 3 types of variable length member descriptor structures, all of which contain the offset required to locate the variable length data. The 'offset' is considered relative to the base address of the parent structure of the variable-length member descriptor. Extra information concerning the variable length data may also be available.

Note that it is not possible to determine whether a variable length data has been initialized before it is referenced. As a convention, the offset plus length equal 0 could be used to indicate an uninitialized variable length data item.

UMAArraryDescr Field - offset

offset (uint32)
Offset to beginning of data.

UMAArraryDescr Field - count

count (uint32)
Count of elements in the array.

UMAArraryDescr Field - size

size (uint32)
Size of each element in the array.

UMADataType

Fundamental data types. May be one of the following data types:

1	UMA_INT32
2	UMA_INT64
3	UMA_UINT32
4	UMA_UINT64
5	UMA_BOOLEAN
6	UMA_OCTETSTRING
7	UMA_TEXTSTRING
8	UMA_TIMEVAL
9	UMA_TIMESPEC
11	UMA_TIMESTAMP

```
typedef uint32 UMADataType;
```

UMAElementDescr

Descriptor for variable-length element (which doesn't contain its own size).

```
typedef struct _UMAElementDescr {
    uint32    offset; /* Offset to beginning of data. */
    uint32    size;   /* Size of the whole structure. */
} UMAElementDescr;

typedef UMAElementDescr *UMAElementDescr;
```

Support for random access to variable length members of structures requires that the address of these members is derived from fixed size structures at known offsets within the main structure definition. There are 3 types of variable length member descriptor structures, all of which contain the offset required to locate the variable length data. The 'offset' is considered relative to the base address of the parent structure of the variable-length member descriptor. Extra information concerning the variable length data may also be available.

Note that it is not possible to determine whether a variable length data has been initialized before it is referenced. As a convention, the offset plus length equal 0 could be used to indicate an uninitialized variable length data item.

UMAElementDescr Field - offset

offset (uint32)
Offset to beginning of data.

UMAElementDescr Field - size

size (uint32)
Size of the whole structure.

UMATextDescr

Descriptor for variable-length text.

```
typedef struct _UMATextDescr {
    uint32    offset; /* Offset to beginning of data. */
    uint32    count;  /* Count of elements in the text. */
} UMATextDescr;

typedef UMATextDescr *UMATextDescr;
```

Support for random access to variable-length members of structures requires that the address of these members is derived from fixed size structures at known offsets within the main structure definition. There are 3 types of variable length member descriptor structures, all of which contain the offset required to locate the variable-length data. The 'offset' is considered relative to the base address of the parent structure of the variable-length member descriptor. Extra information concerning the variable length data may also be available.

Note that it is not possible to determine whether a variable-length data has been initialized before it is referenced. As a convention, setting the offset and count to 0 could be used to indicate an uninitialized variable-length data item.

UMATextDescr Field - offset

offset (uint32)
Offset to beginning of data.

UMATextDescr Field - count

count (uint32)
Count of elements in the text.

UMATextString

Structure for a variable-length string.

```
typedef struct _UMATextString {  
    uint32    size;        /* Size of the entire structure. */  
    int8      string[1];   /* The variable-length string. */  
} UMATextString;  
  
typedef UMATextString *UMATextString;
```

UMATextString Field - size

size (uint32)
Size of the entire structure.

UMATextString Field - string[1]

string[1] (int8)
The variable-length string.

UMAVarArrayDescr

Descriptor for variable-length array of variable-sized elements

```
typedef struct _UMAVarArrayDescr {
    uint32     offset; /* Offset to beginning of data. */
    uint32     count;  /* Count of elements in the array. */
} UMAVarArrayDescr;

typedef UMAVarArrayDescr *UMAArraryDescr;
```

Support for random access to variable length members of structures requires that the address of these members is derived from fixed size structures at known offsets within the main structure definition. There are 3 types of variable length member descriptor structures, all of which contain the offset required to locate the variable length data. The 'offset' is considered relative to the base address of the parent structure of the variable-length member descriptor. Extra information concerning the variable length data may also be available.

Note that it is not possible to determine whether a variable length data has been initialized before it is referenced. As a convention, the offset plus length equal 0 could be used to indicate an uninitialized variable length data item.

UMAVarArrayDescr Field - offset

offset (uint32)
Offset to beginning of data.

UMAVarArrayDescr Field - count

count (uint32)
Count of elements in the array.

UMAVarLenData

Used as a place holder for variable-length data in structures.

```
typedef uint8 UMAVarLenData;
```

UMAVarLenDescr

Descriptor for a single variable-length element which contains its own size

```
typedef struct _UMAVarLenDescr {
    uint32     offset; /* Offset to beginning of data. */
} UMAVarLenDescr;

typedef UMAVarLenDescr *UMAVarLenDescr;
```

Support for random access to variable length members of structures requires that the address of these members is derived from fixed size structures at known offsets within the main structure definition. There are 3 types of variable length member descriptor structures, all of which contain the offset required to locate the variable length data. The 'offset' is considered relative to the base address of the parent structure of the variable-length member descriptor. Extra information concerning the variable length data may also be available.

Note that it is not possible to determine whether a variable length data has been initialized before it is referenced. As a convention, setting the offset to 0 could be used to indicate an uninitialized variable length data item.

UMAVarLenDescr Field - offset

offset (uint32)
Offset to beginning of data.

OpenGL 3D Rendering

OpenGL is an industry standard 3D rendering API available on many different workstation and PC platforms. This powerful rendering API is defined by the OpenGL Architectural Review Board (ARB), of which IBM is a founding member. OpenGL conformance tests are also defined by the ARB, which ensures portability of the API across platforms. Software developers using OpenGL can count on equivalent functionality across all platforms. OpenGL defines only a rendering specification; windowing and input are defined by the environment in which OpenGL is run (in our case OS/2 Presentation Manager).

This implementation of OpenGL conforms to the OpenGL 1.0 specification, and passes the OpenGL 1.0 Conformance tests. OpenGL is intended to be an interface to graphics hardware. This means that application writers do not have to code for a particular piece of graphics hardware since they are ensured of a consistent behavior across all graphics hardware which passes the OpenGL conformance tests.

OpenGL on OS/2 does not require special 3D hardware to run. Every OS/2 platform will contain a complete OpenGL pipeline and rasterizer implemented in software. This will provide entry level 3D graphics functionality for a broader base of users. OpenGL performance will scale with CPU power. Higher floating point power will yield better OpenGL performance. Of course, major performance gains can be realized through the use of special 3D hardware when supported under OS/2.

This chapter includes the following sections:

- [OpenGL Functionality](#)
 - [OpenGL Integration into OS/2](#)
 - [OpenGL Windowing and Input Toolkits](#)
 - [OpenGL References](#)
 - [OpenGL Functions](#)
 - [OpenGL Data Types](#)
-

OpenGL Functionality

OpenGL provides functionality for the following:

- A wide assortment of graphical primitives can be drawn
 - Points, Lines, Segments, Loops, Triangles, Quadrangles, Triangle Strips and more
 - Utility library supports NURBS and complex polygons
- Flexible data input
 - Object data can be specified in 2, 3, or 4 dimensions and numerous data formats
 - Allows developer to use immediate or display list mode
- Complex light models
 - Up to 8 lights
 - Spot lights
 - Local or infinite lights
- Complex material models
 - One or two sided
 - Ambient, diffuse and specular components can be set
- Advanced texture mapping support
 - Automatic generation of texture coordinates
 - Two magnification filters
 - Six minification filters including mipmapping
 - Decal, Modulation, or Blending of texture
- Line and Polygon antialiasing
- Fogging effects
- Accumulation buffer can facilitate motion blur and full scene antialiasing effects
- Depth buffer and comparisons available for hidden surface removal

- Alpha buffer and blending operations facilitate transparency and compositing effects
- Stencil buffer can be used for Constructive Solid Geometry (CSG) modeling and shadows
- Dithering allows true color images to be drawn with a small palette
- Double-buffering for smooth animation

OpenGL Integration into OS/2

A small set of calls integrates OpenGL into whatever windowing system it is running in. Since these calls are windowing-system specific, they cannot be the same across all platforms. In OS/2, these calls make up the PGL (Presentation Manager GL) specification. Application programmers use the PGL interface to:

- Create and Prepare an OpenGL Context for Rendering
- Swap a window's front and back buffers (front buffer is displayed)
- Select a Color Palette for a Color Index Context
- Integrate GPI and OpenGL rendering and fonts

PGL provides much of the same functionality as glX in X Windows systems, and the WGL interface in Microsoft systems.

OpenGL contexts are created using [pglCreateContext](#). One parameter to this call is a [VISUALCONFIG](#), which describes what type of ancillary buffers are needed to go with the rendering context. A list of available VISUALCONFIGs can be queried with [pglQueryConfigs](#). When choosing a [VISUALCONFIG](#), you should look for one which contains just the buffers you plan on using. For instance, choose a VISUALCONFIG with accumulation buffers only if you plan to use them. 64-bits deep accumulation buffers allocated for a 400 by 400 window would allocate 1,280,000 bytes! Therefore, you should choose the visual configuration with the least number of buffers to suit your needs in order to minimize the number of buffers allocated. After a context is created, it must be bound to an OS/2 window by calling [pglMakeCurrent](#). When a context is current, that means it can finally be used for rendering.

OpenGL runs as a set of DLLs on top of OS/2:

- The OPENGL.DLL contains all rendering entry points, glu entry points, PGL entry points, and the OpenGL pipeline code. The pipeline code is responsible for object transformation, lighting, culling and clipping. Applications only need to link with OPENGL.DLL to use the OpenGL API.
- The RASTER.DLL contains a rasterizer, which takes vertices and creates fragments. These fragments then go through a series of possible steps including texture mapping, fogging, depth tests (Z-buffering), blending and many more. Some or all of these steps can be performed by 3D hardware, if available. Using [pglSwapBuffers](#) or [glFlush](#) will flush and display rendered image.

OpenGL Windowing and Input Toolkits

OpenGL on OS/2 is supplied with two toolkits which provide simple windowing and input handling. These toolkits are not intended for application development, but for simple programs and demos. They are provided "as-is" on most OpenGL platforms, including OS/2. They are also used in sample code provided in the *OpenGL Programming Guide* published by Addison-Wesley (see [OpenGL References](#)). The AUX toolkit and GLUT toolkit are provided with OpenGL.

OpenGL References

There are many places to learn about OpenGL on all its supported platforms:

- Web Sites:
 - OpenGL WWW site <http://www.sgi.com/Technology/OpenGL/opengl.html>
 - OpenGL at IBM WWW site <http://www.austin.ibm.com/software/OpenGL>
 - OpenGL Published Benchmarks by OPC (OpenGL Performance Characterization)

- USENET newgroup:
 - comp.graphics.api.opengl
- OpenGL Reference Books
 - *OpenGL Reference Manual, The Official Reference Document for OpenGL, Release 1* , published by Addison Wesley, ISBN: 0-201-63276-4
 - *OpenGL Programming Guide* , published by Addison Wesley, ISBN: 0-201-63274-8
 - *OpenGL Specification* available using anonymous ftp from sgi.com in /sgi/opengl/doc

OpenGL Functions

The OpenGL functions described in this section are:

- [pglChooseConfig](#)
- [pglCopyContext](#)
- [pglCreateContext](#)
- [pglDestroyContext](#)
- [pglGetCurrentContext](#)
- [pglGetCurrentWindow](#)
- [pglGrabFrontBitmap](#)
- [pglIsIndirect](#)
- [pglMakeCurrent](#)
- [pglQueryCapability](#)
- [pglQueryConfigs](#)
- [pglReleaseFrontBitmap](#)
- [pglSwapBuffers](#)
- [pglUseFont](#)
- [pglWaitGL](#)
- [pglWaitPM](#)

pglChooseConfig

pglChooseConfig - Syntax

```
#include <pgl.h>

HAB          hab;
int          *attriblist;
PVISUALCONFIG *pVisualConfig;

pVisualConfig = pglChooseConfig(hab, attriblist);
```

pglChooseConfig Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglChooseConfig Parameter - attriblist

attriblist (int *) - input
A list of attribute/value pairs. The last attribute in the list must be None.

The following is a list of valid attributes:

PGL_BUFFER_SIZE	Must be followed by a nonnegative integer that indicates the desired color index buffer size. The smallest index buffer of at least the specified size is preferred. Ignored if PGL_RGBA is asserted.
PGL_LEVEL	Must be followed by an integer buffer-level specification. This specification is honored exactly. Buffer level zero corresponds to the default frame buffer of the display. Buffer level one is the first overlay frame buffer, level two the second overlay frame buffer, and so on. Negative buffer levels correspond to underlay frame buffers.
PGL_RGBA	If present, only RGBA visual configs are considered. Otherwise, Color Index visual configs are considered.
PGL_DOUBLEBUFFER	If present, only double buffered visual configs are considered. Otherwise both single buffered and double buffered visual configs may be considered.
PGL_SINGLEBUFFER	If present, only single buffered visual configs are considered. Otherwise both single buffered and double buffered visual config may be considered.
PGL_STEREO	If present, only stereo visual configs are considered. Otherwise, both monoscopic and stereoscopic visual configs are considered.
PGL_AUX_BUFFERS	Must be followed by a nonnegative integer that indicates the desired number of auxiliary buffers. Visual configs with the smallest number of auxiliary buffers that meets or exceeds the specified number are preferred.
PGL_RED_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available red buffer is preferred. Otherwise, the largest available red buffer of at least the minimum size is preferred.
PGL_GREEN_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available green buffer is preferred. Otherwise, the largest available green buffer of at least the minimum size is preferred.
PGL_BLUE_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available blue buffer is preferred. Otherwise, the largest available blue buffer of at least the minimum size is preferred.
PGL_ALPHA_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available alpha buffer is preferred. Otherwise, the largest available alpha buffer of at least the minimum size is preferred.
PGL_DEPTH_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no depth buffer are preferred. Otherwise, the largest available depth buffer of at least the minimum size is preferred.

PGL_STENCIL_SIZE

Must be followed by a nonnegative integer that indicates the desired number of stencil bitplanes. The smallest stencil buffer of at least the specified size is preferred. If the desired value is zero, visual configs with no stencil buffer are preferred.

PGL_ACCUM_RED_SIZE

Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no red accumulation buffer are preferred. Otherwise, the largest possible red accumulation buffer of at least the minimum size is preferred.

PGL_ACCUM_GREEN_SIZE

Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no green accumulation buffer are preferred. Otherwise, the largest possible green accumulation buffer of at least the minimum size is preferred.

PGL_ACCUM_BLUE_SIZE

Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no blue accumulation buffer are preferred. Otherwise, the largest possible blue accumulation buffer of at least the minimum size is preferred.

PGL_ACCUM_ALPHA_SIZE

Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no alpha accumulation buffer are preferred. Otherwise, the largest possible alpha accumulation buffer of at least the minimum size is preferred.

pglChooseConfig Return Value - pVisualConfig

pVisualConfig (**PVISUALCONFIG ***) - returns

Pointer to a null-terminated array of pointers to VISUALCONFIG structures containing buffer configs. A NULL is returned if no suitable VisualConfig was found.

pglChooseConfig - Parameters

hab (HAB) - input

Handle to anchor block.

attriblist (int *) - input

A list of attribute/value pairs. The last attribute in the list must be None.

The following is a list of valid attributes:

PGL_BUFFER_SIZE

Must be followed by a nonnegative integer that indicates the desired color index buffer size. The smallest index buffer of at least the specified size is preferred. Ignored if PGL_RGBA is asserted.

PGL_LEVEL

Must be followed by an integer buffer-level specification. This specification is honored exactly. Buffer level zero corresponds to the default frame buffer of the display. Buffer level one is the first overlay frame buffer, level two the second overlay frame buffer, and so on. Negative buffer levels correspond to underlay frame buffers.

PGL_RGBA

If present, only RGBA visual configs are considered. Otherwise, Color Index visual configs are considered.

PGL_DOUBLEBUFFER

If present, only double buffered visual configs are considered. Otherwise both single buffered and double buffered visual configs may be considered.

PGL_SINGLEBUFFER

If present, only single buffered visual configs are considered. Otherwise both single buffered and double buffered visual config may be considered.

PGL_STEREO	If present, only stereo visual configs are considered. Otherwise, both monoscopic and stereoscopic visual configs are considered.
PGL_AUX_BUFFERS	Must be followed by a nonnegative integer that indicates the desired number of auxiliary buffers. Visual configs with the smallest number of auxiliary buffers that meets or exceeds the specified number are preferred.
PGL_RED_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available red buffer is preferred. Otherwise, the largest available red buffer of at least the minimum size is preferred.
PGL_GREEN_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available green buffer is preferred. Otherwise, the largest available green buffer of at least the minimum size is preferred.
PGL_BLUE_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available blue buffer is preferred. Otherwise, the largest available blue buffer of at least the minimum size is preferred.
PGL_ALPHA_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available alpha buffer is preferred. Otherwise, the largest available alpha buffer of at least the minimum size is preferred.
PGL_DEPTH_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no depth buffer are preferred. Otherwise, the largest available depth buffer of at least the minimum size is preferred.
PGL_STENCIL_SIZE	Must be followed by a nonnegative integer that indicates the desired number of stencil bitplanes. The smallest stencil buffer of at least the specified size is preferred. If the desired value is zero, visual configs with no stencil buffer are preferred.
PGL_ACCUM_RED_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no red accumulation buffer are preferred. Otherwise, the largest possible red accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_GREEN_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no green accumulation buffer are preferred. Otherwise, the largest possible green accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_BLUE_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no blue accumulation buffer are preferred. Otherwise, the largest possible blue accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_ALPHA_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no alpha accumulation buffer are preferred. Otherwise, the largest possible alpha accumulation buffer of at least the minimum size is preferred.

pVisualConfig (PVISUALCONFIG *) - returns

Pointer to a null-terminated array of pointers to VISUALCONFIG structures containing buffer configs. A NULL is returned if no suitable VisualConfig was found.

pglChooseConfig - Remarks

Attriblist example:

```
attribList = {PGL_RGBA, PGL_RED_SIZE, 4, PGL_GREEN_SIZE, 4, PGL_BLUE_SIZE, 4, None};
```


Specifies a single-buffered RGB visual config in the normal frame buffer, not an overlay or underlay buffer. The returned visual config supports at least four bits each of red, green, and blue, and possibly no bits of alpha. It does not support color index mode, double-buffering, or stereo display. It may or may not have one or more auxiliary color buffers, a depth buffer, a stencil buffer, or an accumulation buffer. The user should not modify the fields of the returned [VISUALCONFIG](#) structure. When the user is done with the PVISUALCONFIG returned, the memory can be freed by calling the C library routine `free()`.

pglChooseConfig - Related Functions

- [pglQueryConfigs](#)
 - [pglCreateContext](#)
-

pglChooseConfig - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglCopyContext

pglCopyContext - Syntax

This call will copy some portion of state from *hgc_src* to *hgc_dst* . The *attrib_mask* parameter determines what group(s) of state variables will be copied. *attrib_mask* must contain the bitwise OR of the same symbolic names that can be passed to `glPushAttrib`. Set *attrib_mask* to `GL_ALL_ATTRIB_BITS` to copy the max amount of state. This copy can be done only if *hgc_src* and *hgc_dst* were created in the same process.

```
#include <pgl.h>
```

```
HAB      hab;  
HGC      hgc_src;  
HGC      hgc_dst;  
GLuint   attrib_mask;  
BOOL     rc;
```

```
rc = pglCopyContext(hab, hgc_src, hgc_dst,  
                   attrib_mask);
```

pglCopyContext Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglCopyContext Parameter - hgc_src

hgc_src (HGC) - input
Handle to the source OpenGL context.

pglCopyContext Parameter - hgc_dst

hgc_dst (HGC) - input
Handle to the destination OpenGL context.

pglCopyContext Parameter - attrib_mask

attrib_mask (GLuint) - input
Specifies which portions of hgc_src are to be copied to hgc_dst.

pglCopyContext Return Value - rc

rc (BOOL) - returns
The following values may be returned:

TRUE	Context was successfully copied from hgc_src to hgc_dst.
FALSE	Error occurred.

pglCopyContext - Parameters

hab (HAB) - input
Handle to anchor block.

hgc_src (HGC) - input
Handle to the source OpenGL context.

hgc_dst (HGC) - input
Handle to the destination OpenGL context.

attrib_mask (GLuint) - input

Specifies which portions of `hgc_src` are to be copied to `hgc_dst`.

rc (BOOL) - returns

The following values may be returned:

TRUE
FALSE

Context was successfully copied from `hgc_src` to `hgc_dst`.
Error occurred.

pglCopyContext - Remarks

If `hgc_src` is not current to the thread issuing the request, then the state of `hgc_dst` is undefined. Not all values of OpenGL state can be copied. For example, pixel pack and pixel unpack state, render mode state, select and feedback state are not copied. The state that can be copied by this command is exactly the same state that can be manipulated by the OpenGL command `glPushAttrib`.

pglCopyContext - Related Functions

- [pglCreateContext](#)

pglCopyContext - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglCreateContext

pglCreateContext - Syntax

This call will create an OpenGL context using the given visual config. A visual config specifies what framebuffer resources are available to the rendering context. If an OpenGL context was successfully created, a handle to it will be returned, else NULL will be returned.

```
#include <pgl.h>
```

```
HAB          hab;  
PVISUALCONFIG pVisualConfig;  
HGC          ShareList;  
BOOL         IsDirect;  
HGC          hgc;
```

```
hgc = pglCreateContext(hab, pVisualConfig,  
                      ShareList, IsDirect);
```

pglCreateContext Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglCreateContext Parameter - pVisualConfig

pVisualConfig ([PVISUALCONFIG](#)) - input
Pointer to VISUALCONFIG structure containing desired buffer config.

pglCreateContext Parameter - ShareList

ShareList (HGC) - input
Handle of OpenGL context with which to share display lists within a process.

pglCreateContext Parameter - IsDirect

IsDirect (BOOL) - input

TRUE	Bypass GPI, and render to a PM window. Generally faster.
FALSE	This context will use Gpi for OpenGL rendering, using GpiBitBlt to blit OpenGL rendering to a PM window

pglCreateContext Return Value - hgc

hgc (HGC) - returns
Handle to successfully created context, or NULL if an error occurred.

pglCreateContext - Parameters

hab (HAB) - input
Handle to anchor block.

pVisualConfig ([PVISUALCONFIG](#)) - input
Pointer to VISUALCONFIG structure containing desired buffer config.

ShareList (HGC) - input
Handle of OpenGL context with which to share display lists within a process.

IsDirect (BOOL) - input

TRUE Bypass GPI, and render to a PM window. Generally faster.

FALSE This context will use Gpi for OpenGL rendering, using GpiBitBlt to blit OpenGL rendering to a PM window

hgc (HGC) - returns
Handle to successfully created context, or NULL if an error occurred.

pglCreateContext - Remarks

Configurations can be listed through [pglQueryConfigs](#).

Direct context- bypasses PM when displaying OpenGL rendering. Not available on all OS/2 configurations. If IsDirect parameter is TRUE, but a direct context is not available, an indirect context will be created. [pglIsIndirect](#) can be called to see if an OpenGL context is direct or indirect.

Indirect context - uses GpiBitBlt to get image to screen. Allows access to a PM bitmap containing OpenGL rendered image. The PM Bitmap is not guaranteed to contain any OpenGL rendering until the OpenGL graphics pipeline has been flushed ([glFlush](#) , [pglSwapBuffers](#) or [pglWaitGL](#)). If ShareList is not NULL, then all display list definitions are shared by ShareList and the newly created context. An arbitrary number of contexts can share display lists, but each context must be owned by the same process.

pglCreateContext - Related Functions

- [pglQueryConfigs](#)
- [pglChooseConfig](#)
- [pglMakeCurrent](#)
- [pglDestroyContext](#)

pglCreateContext - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglDestroyContext

pglDestroyContext - Syntax

This function will destroy the hgc, and all the resources that belong to it. If it is currently bound to a hwnd, this call will be ignored.

```
#include <pgl.h>

HAB      hab;
HGC      hgc;
BOOL     rc;

rc = pglDestroyContext(hab, hgc);
```

pglDestroyContext Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglDestroyContext Parameter - hgc

hgc (HGC) - input
Handle to an OpenGL context.

pglDestroyContext Return Value - rc

rc (BOOL) - returns
The following values may be returned:

TRUE	Context was successfully destroyed, or context was bound to a window.
FALSE	Error occurred.

If hgc is currently bound to a window, pglDestroyContext will return without destroying hgc. A current context can be unbound from the currently bound window by calling [pglMakeCurrent](#) (hab,NULL,None). [pglMakeCurrent](#)(hab,NULL,None).

pglDestroyContext - Parameters

hab (HAB) - input
Handle to anchor block.

hgc (HGC) - input
Handle to an OpenGL context.

rc (BOOL) - returns

The following values may be returned:

TRUE	Context was successfully destroyed, or context was bound to a window.
FALSE	Error occurred.

If hgc is currently bound to a window, pglDestroyContext will return without destroying hgc. A current context can be unbound from the currently bound window by calling [pglMakeCurrent](#) (hab,NULL,None). [pglMakeCurrent](#)(hab,NULL,None).

pglDestroyContext - Related Functions

- [pglCreateContext](#)
- [pglMakeCurrent](#)

pglDestroyContext - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Related Functions](#)
[Glossary](#)

pglGetCurrentContext

pglGetCurrentContext - Syntax

Look up the current OpenGL context for this process. A context is 'current' if it was the last context to be bound to a window by calling [pglMakeCurrent](#).

```
#include <pgl.h>
```

```
HAB      hab;  
HGC      hgc;
```

```
hgc = pglGetCurrentContext (hab) ;
```

pglGetCurrentContext Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglGetCurrentContext Return Value - hgc

hgc (HGC) - returns

Handle to the current context. NULL is returned if no current context exists for this process.

pglGetCurrentContext - Parameters

hab (HAB) - input

Handle to anchor block.

hgc (HGC) - returns

Handle to the current context. NULL is returned if no current context exists for this process.

pglGetCurrentContext - Remarks

Only One current context is allowed per process.

pglGetCurrentContext - Related Functions

- [pglMakeCurrent](#)
 - [pglCreateContext](#)
 - [pglGetCurrentWindow](#)
-

pglGetCurrentContext - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

pglGetCurrentWindow

pglGetCurrentWindow - Syntax

This function returns the current hwnd that is bound to an OpenGL context. A window is current if it was the last window to be bound to an OpenGL context by calling [pglMakeCurrent](#).

```
#include <pgl.h>

HAB      hab;
HWND     hwnd;

hwnd = pglGetCurrentWindow(hab);
```

pglGetCurrentWindow Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglGetCurrentWindow Return Value - hwnd

hwnd (HWND) - returns
Handle to the current window that is bound to an OpenGL context. NULL is returned if no window is currently bound to any OpenGL context for the calling process.

pglGetCurrentWindow - Parameters

hab (HAB) - input
Handle to anchor block.

hwnd (HWND) - returns
Handle to the current window that is bound to an OpenGL context. NULL is returned if no window is currently bound to any OpenGL context for the calling process.

pglGetCurrentWindow - Remarks

Only one current window is allowed per process.

pglGetCurrentWindow - Related Functions

- [pglMakeCurrent](#)
- [pglGetCurrentContext](#)

pglGetCurrentWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglGrapFrontBitmap

pglGrapFrontBitmap - Syntax

This subroutine queries the HPS and HBITMAP which contain the bitmap representation of the front buffer of the current OpenGL window. While the bitmap is locked, the application will NOT receive WM_SIZE or WM_ADJUSTPOSITION messages in the current OpenGL hwnd, and the current window will NOT be sizeable. An implicit glFlush() occurs before this call completes. Applications must call [pglReleaseFrontBitmap](#) when they are done with the HBITMAP. The HBITMAP is only valid between pglGrabFrontBitmap and pglReleaseFrontBitmap, the HPS will be valid until the current window is unbound by calling pglMakeCurrent.

```
#include <pgl.h>

HAB      hab;
HPS      *phps;
HBITMAP  *phbitmap;
BOOL     rc;

rc = pglGrapFrontBitmap(hab, phps, phbitmap);
```

pglGrapFrontBitmap Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglGrapFrontBitmap Parameter - phps

phps (HPS *) - input
Pointer returning HPS with phbitmap set in it.

pglGrapFrontBitmap Parameter - phbitmap

phbitmap (HBITMAP *) - input
Pointer returning HBITMAP containing OpenGL rendering.

pglGrapFrontBitmap Return Value - rc

rc (BOOL) - returns

TRUE	Bitmap was grabbed
FALSE	No Current window exists for this process

pglGrapFrontBitmap - Parameters

hab (HAB) - input
Handle to anchor block.

phps (HPS *) - input
Pointer returning HPS with phbitmap set in it.

phbitmap (HBITMAP *) - input
Pointer returning HBITMAP containing OpenGL rendering.

rc (BOOL) - returns

TRUE	Bitmap was grabbed
FALSE	No Current window exists for this process

pglGrapFrontBitmap - Remarks

Applications should not destroy the HPS or HBITMAP. Applications should not use GpiSetPalette to modify the HPS's color palette. Applications should not disassociate HPS from it's DC, or unset the bitmap from HPS. Applications should not set any PS size, units and format using GpiSetPS. In other words, "Look, but don't touch!" See [pglWaitPM](#) and [pglWaitGL](#) for integrating OpenGL and Gpi drawing. Applications should call [pglReleaseFrontBitmap](#) as soon as possible after locking the bitmap with this call.

pglGrapFrontBitmap - Related Functions

- [pglIsIndirect](#)
- [pglReleaseFrontBitmap](#)
- [pglWaitGL](#)
- [pglWaitPM](#)

pglGrapFrontBitmap - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

pgllsIndirect

pgllsIndirect - Syntax

This function will return what type of context hgc is. The hgc parameter must have been returned from [pglCreateContext](#).

```
#include <pgl.h>
```

```
HAB      hab;  
HGC      hgc;  
LONG     rc;
```

```
rc = pglIsIndirect(hab, hgc);
```

pgllsIndirect Parameter - hab

hab (HAB) - input
Handle to anchor block.

pgllsIndirect Parameter - hgc

hgc (HGC) - input
Handle to an OpenGL context.

pgllsIndirect Return Value - rc

rc (LONG) - returns

The following values may be returned:

0
< 0
1

This Context bypasses conventional PM blitting methods, and is faster.

Error occurred.

This Context uses conventional PM blitting methods, and while it is slower, allows an application to integrate OpenGL rendering and Gpi rendering commands. See Remarks section.

pgllsIndirect - Parameters

hab (HAB) - input

Handle to anchor block.

hgc (HGC) - input

Handle to an OpenGL context.

rc (LONG) - returns

The following values may be returned:

0
< 0
1

This Context bypasses conventional PM blitting methods, and is faster.

Error occurred.

This Context uses conventional PM blitting methods, and while it is slower, allows an application to integrate OpenGL rendering and Gpi rendering commands. See Remarks section.

pgllsIndirect - Remarks

If pgllsIndirect returns 1, it uses Gpi to blit OpenGL rendering. Integration of OpenGL rendering and Gpi rendering is only allowed on Indirect contexts, and is controlled through the use of [pglWaitPM](#) and [pglWaitGL](#). Applications must call [pglGrapFrontBitmap](#) when they need access to the actual bitmap, and call [pglReleaseFrontBitmap](#) when they are done with it. The PM Bitmap is not guaranteed to contain any OpenGL rendering until the OpenGL graphics pipeline has been flushed ([glFlush](#), [pglSwapBuffers](#), or [pglWaitGL](#)).

pgllsIndirect - Related Functions

- [pglCreateContext](#)
-

pgllsIndirect - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

pglMakeCurrent

pglMakeCurrent - Syntax

```
#include <pgl.h>

HAB      hab;
HGC      hgc;
HWND     hwnd;
BOOL     rc;

rc = pglMakeCurrent(hab, hgc, hwnd);
```

pglMakeCurrent Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglMakeCurrent Parameter - hgc

hgc (HGC) - input
Handle to an OpenGL context.

pglMakeCurrent Parameter - hwnd

hwnd (HWND) - input
Handle to a PM window.

pglMakeCurrent Return Value - rc

rc (BOOL) - returns
The following values may be returned:

TRUE
FALSE

Context was successfully destroyed, or context was bound to a window.
Error occurred.

pglMakeCurrent - Parameters

hab (HAB) - input
Handle to anchor block.

hgc (HGC) - input
Handle to an OpenGL context.

hwnd (HWND) - input
Handle to a PM window.

rc (BOOL) - returns
The following values may be returned:

TRUE
FALSE

Context was successfully destroyed, or context was bound to a window.
Error occurred.

pglMakeCurrent - Remarks

Only one context can be bound to a window at a time. The PM window must have been created with window client class CS_SIZEREDRAW and CS_MOVENOTIFY. The application cannot sub-class the window with WinSubclassWindow(hwnd) while an OpenGL context is bound to hwnd.

pglMakeCurrent - Related Functions

- [pglCreateContext](#)
- [pglGetCurrentContext](#)
- [pglGetCurrentWindow](#)

pglMakeCurrent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglQueryCapability

pglQueryCapability - Syntax

Query OpenGL capability on this machine.

```
#include <pgl.h>

HAB      hab;
LONG     rc;

rc = pglQueryCapability(hab);
```

pglQueryCapability Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglQueryCapability Return Value - rc

rc (LONG) - returns

0	No OpenGL Support on this machine
1	OpenGL Support available through PM blitting only
2	Advanced OpenGL support available

pglQueryCapability - Parameters

hab (HAB) - input
Handle to anchor block.

rc (LONG) - returns

0	No OpenGL Support on this machine
1	OpenGL Support available through PM blitting only
2	Advanced OpenGL support available

pglQueryCapability - Topics

Select an item:

[Syntax](#)
[Parameters](#)

pglQueryConfigs

pglQueryConfigs - Syntax

This call will return a NULL terminated array of pointers to available visual config structures for an OpenGL application to choose from. A visual config defines what buffer configurations (depth,accum,alpha,stencil) are available.

```
#include <pgl.h>

HAB          hab;
PVISUALCONFIG pVisualConfig;

pVisualConfig = pglQueryConfigs(hab);
```

pglQueryConfigs Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglQueryConfigs Return Value - pVisualConfig

pVisualConfig ([PVISUALCONFIG](#)) - returns
A NULL-terminated array of pointers to VisualConfig structures. A NULL is returned if no suitable VisualConfig was found.

pglQueryConfigs - Parameters

hab (HAB) - input
Handle to anchor block.

pVisualConfig ([PVISUALCONFIG](#)) - returns
A NULL-terminated array of pointers to VisualConfig structures. A NULL is returned if no suitable VisualConfig was found.

pglQueryConfigs - Remarks

An application should choose the simplest visual config that will suit its needs (least number of buffers). Programmers can also use the helper routine [pglChooseConfig](#), which will choose a suitable visual based on minimum requirements. A [VISUALCONFIG](#) is required to create an OpenGL context. The list of Visual Configs is not guaranteed to be in any certain order.

Note: The user should not modify the fields of the returned PVISUALCONFIG structures. When the user is done with the returned PVISUALCONFIG list, that memory can be freed by calling the C library routine `free()`.

pglQueryConfigs - Related Functions

- [pglChooseConfig](#)
- [pglCreateContext](#)

pglQueryConfigs - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglReleaseFrontBitmap

pglReleaseFrontBitmap - Syntax

This subroutine releases the HBITMAP which was grabbed using [pglGrapFrontBitmap](#). The calling process will now begin receiving WM_SIZE and WM_ADJUSTPOSITION messages in its current OpenGL window again.

```
#include <pgl.h>

HAB      hab;
BOOL     rc;

rc = pglReleaseFrontBitmap(hab);
```

pglReleaseFrontBitmap Parameter - hab

hab (HAB) - input

Handle to anchor block.

pglReleaseFrontBitmap Return Value - rc

rc (BOOL) - returns

TRUE	Bitmap was released
FALSE	No bitmap needed to be released

pglReleaseFrontBitmap - Parameters

hab (HAB) - input
Handle to anchor block.

rc (BOOL) - returns

TRUE	Bitmap was released
FALSE	No bitmap needed to be released

pglReleaseFrontBitmap - Remarks

See [pglWaitPM](#) and [pglWaitGL](#) for integrating OpenGL and PM drawing. Applications should call [pglReleaseFrontBitmap](#) as soon as possible after locking the bitmap with [pglGrapFrontBitmap](#).

pglReleaseFrontBitmap - Related Functions

- [pglIsIndirect](#)
- [pglGrapFrontBitmap](#)
- [pglWaitGL](#)
- [pglWaitPM](#)

pglReleaseFrontBitmap - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglSelectColorIndexPalette

pglSelectColorIndexPalette - Syntax

pglSelectColorIndexPalette passes in a color index palette for OpenGL to use. This function is only needed when using a context with a color index [VISUALCONFIG](#). When using an RGB [VISUALCONFIG](#), OpenGL will set up and select the palette itself when needed. pglSelectColorIndexPalette must be made before a context can be bound a window for the first time.

```
#include <pgl.h>

HAB      hab;    /* Handle to the anchor block. */
HPAL     hpal;   /* Handle to the palette. */
HGC      hgc;    /* Handle to the color index OpenGL context. */
BOOL     rc;     /* Success indicator. */

rc = pglSelectColorIndexPalette(hab, hpal,
                                hgc);
```

pglSelectColorIndexPalette Parameter - hab

hab (HAB) - input
Handle to the anchor block.

pglSelectColorIndexPalette Parameter - hpal

hpal (HPAL) - input
Handle to the palette.

pglSelectColorIndexPalette Parameter - hgc

hgc (HGC) - input
Handle to the color index OpenGL context.

pglSelectColorIndexPalette Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE
FALSE

Palette selection was successful.
An error occurred.

pglSelectColorIndexPalette - Parameters

hab (HAB) - input
Handle to the anchor block.

hpal (HPAL) - input
Handle to the palette.

hgc (HGC) - input
Handle to the color index OpenGL context.

rc (BOOL) - returns
Success indicator.

TRUE
FALSE

Palette selection was successful.
An error occurred.

pglSelectColorIndexPalette - Remarks

You should NOT call GpiSelectPalette; OpenGL will do this for you. pglSelectColorIndexPalette is only used for Color Index contexts.

pglSelectColorIndexPalette - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

pglSwapBuffers

pglSwapBuffers - Syntax

Swaps the front and back buffers of hwnd. This routine has no effect on windows attached to single buffered contexts. An implicit glFlush is done by pglSwapBuffers before it returns. Double buffered rendering is done when smooth animation between frames is desired. OpenGL

commands issued after calling pglSwapBuffers are not issued until the buffer swap is complete.

```
#include <pgl.h>
```

```
HAB      hab;  
HWND     hwnd;
```

```
pglSwapBuffers(hab, hwnd);
```

pglSwapBuffers Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglSwapBuffers Parameter - hwnd

hwnd (HWND) - input
Handle to a PM window.

pglSwapBuffers - Return Value

pglSwapBuffers - Parameters

hab (HAB) - input
Handle to anchor block.

hwnd (HWND) - input
Handle to a PM window.

pglSwapBuffers - Remarks

When a window's buffers are swapped, the back buffer becomes the front, and the front buffer will become the back. The programmer can control which buffer is affected by OpenGL rendering calls through the use of glDrawBuffer. Front and back buffers are not created for a window until it has been bound to an OpenGL context. This call has no effect on a PM window which has never been bound to an OpenGL

context. The window specified by *hwnd* does not CURRENTLY have to be bound to an OpenGL context, just needs to have been bound at some point.

pglSwapBuffers - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

pglUseFont

pglUseFont - Syntax

This function will create count OpenGL display lists containing bitmaps of the named logical character set identifier (*llcid*) specified. An OS/2 logical font should be created by the user using the specified *hps*, *llcid*, and *fattrs* before calling *pglUseFont*. Each bitmap will consist of a single *glBitmap* command. These display lists will be numbered *listbase*, through *listbase* + count -1. The parameters to *glBitmap* for display list *listbase* +i are derived from bitmap *first* +i in the logical font. OpenGL might delay *glBitmap* creation until a font glyph is accessed.

```
#include <pgl.h>

HAB      hab;
HPS      hps;
FATTRS   fattrs;
LONG     llcid;
int      first;
int      count;
int      listbase;
BOOL     rc;

rc = pglUseFont(hab, hps, fattrs, llcid, first,
               count, listbase);
```

pglUseFont Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglUseFont Parameter - hps

hps (HPS) - input

Handle to a PS that the logical font was created with.

pglUseFont Parameter - fattrs

fattrs (**FATTRS**) - input
Font Attributes (was used in GpiCreateLogFont).

pglUseFont Parameter - lcid

lcid (**LONG**) - input
Logical Set ID for created Logical Font.

pglUseFont Parameter - first

first (**int**) - input
Index of first glyph to be taken.

pglUseFont Parameter - count

count (**int**) - input
Number of glyphs to be taken.

pglUseFont Parameter - listbase

listbase (**int**) - input
Index of first display list to be created.

pglUseFont Return Value - rc

rc (**BOOL**) - returns

FALSE

Error occurred.

TRUE

Bitmap display lists were successfully created.

pglUseFont - Parameters

hab (HAB) - input
Handle to anchor block.

hps (HPS) - input
Handle to a PS that the logical font was created with.

fattrs ([FATTRS](#)) - input
Font Attributes (was used in GpiCreateLogFont).

lloid (LONG) - input
Logical Set ID for created Logical Font.

first (int) - input
Index of first glyph to be taken.

count (int) - input
Number of glyphs to be taken.

listbase (int) - input
Index of first display list to be created.

rc (BOOL) - returns

FALSE
TRUE

Error occurred.
Bitmap display lists were successfully created.

pglUseFont - Remarks

Empty display lists are created for all glyphs requested but not defined in the logical font specified by id. No display lists will be created if there is no current OpenGL context.

pglUseFont - Related Functions

- [pglIsIndirect](#)
- [pglWaitGL](#)
- [pglWaitPM](#)

pglUseFont - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)

pglWaitGL

pglWaitGL - Syntax

Ensure that all OpenGL rendering commands made prior to pglWaitGL are executed before and Gpi rendering calls made after pglWaitGL. This call is ignored if there is no current context or if the current context is a direct context which does not use Gpi for displaying OpenGL images. The return value for this function is HPS that has the bitmap set in it to be the bitmap containing OpenGL rendering.

```
#include <pgl.h>

HAB      hab;
HPS      hps;

hps = pglWaitGL(hab);
```

pglWaitGL Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglWaitGL Return Value - hps

hps (HPS) - returns
Handle to the PS which is used to blit the OpenGL image to the screen. This HPS can be used for Gpi drawing commands. NULL is returned if no Current Context exists, or current context is not using Gpi to display OpenGL images.

pglWaitGL - Parameters

hab (HAB) - input
Handle to anchor block.

hps (HPS) - returns
Handle to the PS which is used to blit the OpenGL image to the screen. This HPS can be used for Gpi drawing commands. NULL is returned if no Current Context exists, or current context is not using Gpi to display OpenGL images.

pglWaitGL - Remarks

Gpi drawing intermingled with OpenGL drawing is only allowed on Indirect contexts that use Gpi to blit their OpenGL rendering to the screen. Prior to doing any Gpi drawing, the user must call pglWaitGL in order to flush the OpenGL rendering stream. pglWaitGL is ignored if there is no current context.

pglWaitGL - Related Functions

- [pglIsIndirect](#)
 - [pglWaitPM](#)
-

pglWaitGL - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

pglWaitPM

pglWaitPM - Syntax

Gpi calls made prior to pglWaitPM are guaranteed to be executed before OpenGL rendering calls made after pglWaitPM.

```
#include <pgl.h>

HAB      hab;

pglWaitPM(hab);
```

pglWaitPM Parameter - hab

hab (HAB) - input
Handle to anchor block.

pglWaitPM - Return Value

pglWaitPM - Parameters

hab (HAB) - input
Handle to anchor block.

pglWaitPM - Remarks

Gpi drawing intermingled with OpenGL drawing is only allowed on Indirect contexts that use Gpi to blit their OpenGL rendering to the screen. Gpi drawing is not guaranteed to be visible until pglWaitPM has been called. Gpi has no concept of front and back buffers. All Gpi drawing commands are therefore assumed to affect the OpenGL front buffer. pglWaitPM is ignored if there is no current context.

pglWaitPM - Related Functions

- [pglIsIndirect](#)
- [pglWaitGL](#)

pglWaitPM - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

OpenGL Data Types

This section describes the following data types that are used with the pgl functions:

VISUALCONFIG

The Visual Configuration structure is required when creating an OpenGL context.

```
typedef struct _VISUALCONFIG {
    ULONG          vid;           /* Visual ID. */
    BOOL           rgba;
    int            redSize;
    int            greenSize;
    int            blueSize;
    int            alphaSize;
    ULONG          redMask;
    ULONG          greenMask;
    ULONG          blueMask;
    ULONG          accumRedSize;
    ULONG          accumGreenSize;
    ULONG          accumBlueSize;
    ULONG          accumAlphaSize;
    BOOL           doubleBuffer;
    BOOL           stereo;
    int            bufferSize;
    int            depthSize;
    int            stencilSize;
    int            auxBuffers;
    int            level;
    PVOID          reserved;
    struct visualconfig *next;
} VISUALCONFIG;

typedef VISUALCONFIG *PVISUALCONFIG;
```

VISUALCONFIG Field - vid

vid (ULONG)
Visual ID.

VISUALCONFIG Field - rgba

rgba (BOOL)

VISUALCONFIG Field - redSize

redSize (int)

VISUALCONFIG Field - greenSize

greenSize (int)

VISUALCONFIG Field - blueSize

blueSize (int)

VISUALCONFIG Field - alphaSize

alphaSize (int)

VISUALCONFIG Field - redMask

redMask (ULONG)

VISUALCONFIG Field - greenMask

greenMask (ULONG)

VISUALCONFIG Field - blueMask

blueMask (ULONG)

VISUALCONFIG Field - accumRedSize

accumRedSize (ULONG)

VISUALCONFIG Field - accumGreenSize

accumGreenSize (ULONG)

VISUALCONFIG Field - accumBlueSize

accumBlueSize (ULONG)

VISUALCONFIG Field - accumAlphaSize

accumAlphaSize (ULONG)

VISUALCONFIG Field - doubleBuffer

doubleBuffer (BOOL)

VISUALCONFIG Field - stereo

stereo (BOOL)

VISUALCONFIG Field - bufferSize

bufferSize (int)

VISUALCONFIG Field - depthSize

depthSize (int)

VISUALCONFIG Field - stencilSize

stencilSize (int)

VISUALCONFIG Field - auxBuffers

auxBuffers (int)

VISUALCONFIG Field - level

level (int)

VISUALCONFIG Field - reserved

reserved (PVOID)

VISUALCONFIG Field - next

next (struct visualconfig *)

OpenGL Sample

The following sample code draws a gouraud shaded octahedron.

```
/* sample code for using OpenGL and PGL */
/* Draws a gouraud shaded octahedron */

#include <stdio.h>
#include "pgl.h" /* PGL calls */
#include "gl.h" /* OpenGL calls */
#define PM_ESCAPE 0x0f
#define MSGBOXID 22
#define SQRT2 1.414

/* attributes passed into pglChooseConfig */
int attriblist[] = {
    PGL_DOUBLEBUFFER, /* request doublebuffered visual config */
    PGL_RGBA, /* request rgb (true color) visual config */
    None /* always end list with this */
};

HAB hab;

void DispError(PSZ errstr)
{
    char buffer[256];
    sprintf(buffer, "Error (0x%x) in SAMPOGL.EXE:", WinGetLastError(hab));
    WinMessageBox(HWND_DESKTOP, HWND_DESKTOP, errstr, buffer,
        MSGBOXID, MB_MOVEABLE!MB_CUACRITICAL!MB_CANCEL);
    exit(0);
}

void Setup()
{
    glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_FALSE);
    glDepthMask(GL_FALSE);
    glEnable(GL_CULL_FACE);
}

float verts[3][3] = {
    { 0.0, 0.0, (1.0/SQRT2) },
    { 0.5, 0.5, 0.0 },
    { -0.5, 0.5, 0.0 },
    { -0.5, -0.5, 0.0 },
    { 0.5, -0.5, 0.0 },
    { 0.0, 0.0, -(1.0/SQRT2) }
};

float colors[3][3] = {
    { 1.0, 1.0, 1.0 },
    { 1.0, 0.0, 0.0 },
    { 0.0, 1.0, 0.0 },
    { 0.0, 0.0, 1.0 },
    { 1.0, 0.0, 1.0 },
    { 0.0, 1.0, 1.0 },
};

HRESULT WINAPI WindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    static float t = 0.0;
    static SWP clientsize;
    static USHORT mycode;
    static UCHAR key;

    switch(msg) {
    case WM_SIZE:
        /* Upon a resize, query new window size and set OpenGL viewport */
        WinQueryWindowPos(hwnd, &clientsize);
        glViewport(0, 0, clientsize.cx, clientsize.cy);
        return WinDefWindowProc(hwnd, msg, mp1, mp2);
    case WM_TIMER:
        /* Upon getting a timer message, the invalidate rectangle call */
        /* will cause a WM_PAINT message to be sent, enabling animation */
    }
```

```

    WinInvalidateRect(hwnd, NULLHANDLE, NULL);
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
case WM_PAINT:
    /* This is what is done for every frame of the animation */
    t += 1.0;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glRotatef(t, 1.0, 1.0, 1.0);
    glBegin(GL_TRIANGLE_FAN);
    glColor3fv(colorsY0");
    glVertex3fv(vertsY0");
    glColor3fv(colorsY1");
    glVertex3fv(vertsY1");
    glColor3fv(colorsY2");
    glVertex3fv(vertsY2");
    glColor3fv(colorsY3");
    glVertex3fv(vertsY3");
    glColor3fv(colorsY4");
    glVertex3fv(vertsY4");
    glColor3fv(colorsY1");
    glVertex3fv(vertsY1");
    glEnd();
    glBegin(GL_TRIANGLE_FAN);
    glColor3fv(colorsY5");
    glVertex3fv(vertsY5");
    glColor3fv(colorsY1");
    glVertex3fv(vertsY1");
    glColor3fv(colorsY4");
    glVertex3fv(vertsY4");
    glColor3fv(colorsY3");
    glVertex3fv(vertsY3");
    glColor3fv(colorsY2");
    glVertex3fv(vertsY2");
    glColor3fv(colorsY1");
    glVertex3fv(vertsY1");
    glEnd();
    glPopMatrix();
    pglSwapBuffers(hab, hwnd);
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
case WM_CHAR:
    mycode = (USHORT)SHORT1FROMMP(mp1);
    if ((mycode & KC_CHAR) && !(mycode & KC_KEYUP))
        key = CHAR1FROMMP(mp2);
    else if ((mycode & KC_VIRTUALKEY) && !(mycode & KC_KEYUP))
        key = CHAR3FROMMP(mp2);
    if (key == PM_ESCAPE)
        WinPostMsg(hwnd, WM_CLOSE, (MPARAM)0, (MPARAM)0);
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
default:
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
}
}
main(int argc, char **argv)
{
    PVISUALCONFIG vishead; /* visual configuration */
    HMQ hmq; /* message queue */
    HWND hwnd;
    HWND hwndFrame;
    ULONG createflags = FCF_TITLEBAR |
        FCF_SYSMENU |
        FCF_MINMAX |
        FCF_SIZEBORDER;
    QMSG qmsg; /* message */
    HGC hgc; /* OpenGL context */
    int major, minor; /* OpenGL version */
    int err;

    hab = WinInitialize(0);
    hmq = WinCreateMsgQueue(hab, 0);
    if (!hmq)
        DispError("Couldn't create a message queue!\n");

    /* Check to see if OpenGL exists */
    if (pglQueryCapability(hab)) {
        pglQueryVersion(hab, &major, &minor);
        /* Version 1.0 */
        if ((major == 1) && (minor == 0)) {
            /* Choose a visual configuration that matches desired */
            /* attributes in attriblist */
            vishead = pglChooseConfig(hab, attriblist);
            if (!vishead)
                DispError("Couldn't find a visual!\n");
            if (WinRegisterClass(

```

```

        hab,
        (PSZ)"PGLtest",
        WindowProc,
        CS_SIZEREDRAW ! CS_MOVENOTIFY, /* Need at least this! */
        0))
{
    hwndFrame = WinCreateStdWindow (
        HWND_DESKTOP,          /* Child of the desktop */
        WS_VISIBLE,            /* Frame style */
        &createflags,           /* min FCF_MENU!FCF_MINMAX */
        (PSZ)"PGLtest",        /* class name */
        "OpenGL Sample",        /* window title */
        WS_VISIBLE,            /* client style */
        0,                      /* resource handle */
        1,                      /* Resource ID */
        &hwnd);                 /* Window handle */

    if (!hwndFrame)
        DispError("Couldn't create a window!\n");
    /* you must set window size before you call pglMakeCurrent */
    if (!WinSetWindowPos(
        hwndFrame,
        HWND_TOP,
        0,
        0,
        300,
        300,
        SWP_ACTIVATE ! SWP_SIZE ! SWP_MOVE ! SWP_SHOW))
        DispError("Couldn't position window!\n");
    hgc = pglCreateContext(hab, /* anchor block handle */
        vishead,               /* visual configuration */
        (HGC)NULL,             /* (no) shared contexts */
        (BOOL)TRUE);           /* direct (fast) context */
    if (!hgc)
        DispError("Couldn't create an OpenGL context!\n");
    if (!pglMakeCurrent(hab, hgc, hwnd))
        DispError("Could not bind OpenGL context to window!\n");
    /* Don't subclass your window past here! */
    Setup();
    /* Start timer to cause WM_TIMER messages to be sent */
    /* periodically. This is used to animate. */
    WinStartTimer(hab, hwnd, 0L, 0L);
    while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
        WinDispatchMsg(hab, &qmsg);
}
}
}
}

```

OS/2 Character Mode Output

Sending output to the console device (stdout) is implemented by doing a DosWrtTty. DosWrtTty writes a character string to the display starting at the current cursor position. At the completion of the write, the cursor is positioned at the end of the string. Characters are written emulating an ASCII terminal. What control sequences are processed is based on whether the ANSI mode is set. If ANSI is off, only a small set of single byte control characters is used. If ANSI is on, a large set of editing controls are used.

Single Byte Controls

The following output controls are always processed:

Bel (0x07)	Bell
Bsp (0x08)	Backspace (erase previous character)
Tab (0x09)	Horizontal tab
Lf (0x0a)	Line feed
Cr (0x0d)	Carriage return
Cr (0x0d)	Carriage return

ANSI controls

The following controls are processed only if the ANSI option is enabled in the current session. These controls are designed to be ANSI X3.64-1979 compliant, and are based on the AIXTERM and DEC-VT102 controls. They are an extension to the ANSI controls supported in DOS and OS/2 2.x. Those controls labeled (DOS) are also supported in DOS ANSI.SYS.

Where the # is shown, it can be replaced with a number. When a value is not given, it defaults to 1 for counts and 0 for positions and options. Columns and rows are numbered starting at 1. Controls are parsed as specified by ANSI X3.64. In addition, the VT100 syntax (ANSI X3.41-1974) is also allowed. Only the 7-bit control formats can be used. Controls which are not in the list below are parsed and ignored.

Control	Description
Esc (0x1b)	Escape. This character introduces a control sequence. If the following character is less than space, it is placed in the datastream, otherwise it is considered a control sequence.
Esc[#@	Insert character. Insert # characters at the current position. The rest of the line is shifted over n characters.
Esc[#A	Cursor up. Move the cursor up # lines. If at top of the screen do nothing. The default number of lines is one. (DOS)
Esc[#B	Cursor down. Move the cursor down # lines. If at the bottom of the screen do nothing. The default number of lines is one. (DOS)
Esc[#C	Cursor forward. Move the cursor to the right # position. If at the edge of the screen do nothing. The default number of positions is one. (DOS)
Esc[#D	Cursor back. Move the cursor to the left # positions. If at the edge of the screen do nothing. The default number of positions is one. (DOS)
Esc[#E	Cursor next line. Move the cursor to the beginning of the next line or lines. Scroll if necessary.
Esc[#F	Cursor previous line. Move the cursor to the beginning of the previous line or lines. Scroll if necessary.
Esc[#G	Cursor set column. Move the cursor to the # column on the current line. If the current line has fewer columns, move the cursor to the end of the line. The default column is 1.
Esc[#;#H	Cursor set row and column. Set the cursor row and column. The first parameter gives the row, and the second parameter gives the column. The default for each parameter is one. (DOS)
Esc[#I	Cursor horizontal tab. Move the cursor to the next tab stop. The default count or 0 give the next tab stop, a parameter of 1 or greater gives the count of tab stops after that one.
Esc[0J	Erase display from cursor to bottom.
Esc[1J	Erase display from top to cursor.
Esc[2J	Erase entire display. The cursor is moved to the top left corner of the display. (DOS)
Esc[0K	Erase line from cursor to end. (DOS)
Esc[1K	Erase line from start to cursor
Esc[2K	Erase the entire line. The cursor is moved to the beginning of the line.
Esc[#L	Insert line. Insert lines after the current line. This causes line below this to be scrolled down.
Esc[#M	Delete line. Delete lines starting with the current line and going down. This causes the area below this to be scrolled up.
Esc[0O	Erase scroll area from cursor to bottom.
Esc[1O	Erase scroll area from top to cursor.
Esc[2O	Erase entire scroll area. The scroll area is set using Esc[r
Esc[#P	Delete character. Delete the current character. This causes the rest of the line to be moved.
Esc[#S	Scroll up. Scroll up the current scrolling area. Lines at the bottom are erased.
Esc[#T	Scroll down. Scroll down the current scrolling area. Lines at the top are erased.

Esc[#X	Erase character. Erase characters starting at the current position. Leave the cursor position unchanged.
Esc[#Z	Cursor backtab. Move the cursor back the specified number of tabs. The default value of zero is to the previous tab.
Esc[#'	Set absolute horizontal cursor position. The current row is unchanged. This is the same as Esc[#G
Esc[#a	Set relative horizontal cursor position. The current row is unchanged. This is the same as Esc[#C
Esc[c	Request device attributes. This causes the string Esc[=20;2;2c to be placed in the input.
Esc[#d	Set absolute vertical cursor position. The current column is unchanged.
Esc[#e	Set relative vertical cursor position. The current column is unchanged. This is the same as Esc[#B
Esc[#;#f	Set cursor position row and column. This is the same as Esc[H.
Esc[0g	Clear tab at current column.
Esc[3g	Clear all tabs.
Esc[4h	Set insert mode.
Esc[20h	Set linefeed newline mode. Linefeed characters move the cursor to the first position of the next line.
Esc[4l	Set replace mode.
Esc[20l	Reset linefeed newline mode. Linefeed characters move the cursor down one line.
Esc[0m	Set normal text. This is set to the color of the last Esc[=0m. (DOS)
Esc[1m	Set bold text. (DOS)
Esc[4m	Set underscore text. Not all displays support this. (DOS)
Esc[5m	Set blink text. Not all displays support this. (DOS)
Esc[7m	Set reverse text. This is the reverse of the current color. (DOS)
Esc[8m	Set invisible text (DOS)
Esc[10m	Use primary font
Esc[11m	Use alternate font
Esc[#;#m	Set text color. The following colors can be specified. Multiple can be specified together to give both a foreground and background color. (DOS)
	30 Black foreground
	31 Red foreground
	32 Green foreground
	33 Brown foreground
	34 Blue foreground
	35 Magenta foreground
	36 Cyan foreground
	37 White foreground
	40 Black background
	41 Red background
	42 Green background
	43 Brown background
	44 Blue background
	45 Magenta background

46 Cyan background

47 White foreground

50 Gray foreground

51 Light-red foreground

52 Light-green foreground

53 Yellow foreground

54 Light-blue foreground

55 Light-magenta foreground

56 Light-cyan foreground

57 Light-white foreground

Esc[6n	Request cursor position. Place the current cursor position in the input buffer as Esc[<i>#</i> ; <i>#</i> R. (DOS)
Esc[<i>#</i> ; <i>#</i> r	Set top and bottom margins. The first parameter gives the top of the scrolling area, and the second parameter gives the bottom of the scrolling area.
Esc[s	Save cursor. The current cursor location is saved and can be restored with Esc[u]. Only one location is saved. This also saves the cursor visibility and bell. (DOS)
Esc[u	Restore cursor. Set the cursor to the saved location. (DOS)
Esc[0v	Set cursor to underline cursor visible.
Esc[1v	Set cursor invisible.
Esc[2v	Set cursor to block cursor and visible.
Esc[?7h	Set wrap mode. When the end of the line is reached do an automatic new line. For insert and delete characters scroll the rest of the display. (DOS)
Esc[?7l	Set nowrap mode. When the end of the line is reached do not do an automatic new line. For insert and delete characters only affect the current line. (DOS)
Esc[= <i>#</i> ; <i>#</i> B	Set bell tone. The pitch is given as a value indicating frequency in Hertz (0 is quiet). The second value gives the duration in 1/100 seconds. This can be 0 to 250.
Esc[= <i>#</i> ; <i>#</i> ; <i>#</i> D	Set display size. There are three values. The first gives the number of columns, and the second gives the number of rows. The actual size may differ from what is specified. The optional third value gives the attribute format. 0=VGA, 1=Extended, 2=Unicode.
Esc[=3h	Set the display to an 80x25 VGA text mode. This is the equivalent of Esc[=80;25D. (DOS)
Esc[=7h	Set wrap mode. When the end of the line is reached do an automatic new line. (DOS)
Esc[= <i>#</i> ; <i>#</i> k	Set typomatic rate. There are two values. The first gives the repeat rate per second (1-30), and the second gives the delay time in 1/100 second (25-100). The typomatic rate is set for all sessions.
Esc[=3l	Reset mode extended. This is the same as Esc[=3h. (DOS)
Esc[=7l	Set nowrap mode. When the end of the line is reached do not do an automatic new line. (DOS)
Esc[=0m	Make the current color the default text color.
Esc[=9m	Reset the default text color.
Esc7	Save cursor (VT100). Same as Esc[s.
Esc8	Restore cursor (VT100). Same as Esc[u.
Esc=	Numlock off (VT100). Turns off the numlock on the keyboard.
Esc >	Numlock on (VT100). Turns on the numlock on the keyboard.
EscD	Index. Move cursor down one line scrolling if necessary.
EscE	New line. Move the cursor to the beginning of the next line.

EscH	Set horizontal tab. Set a tab at the current cursor position.
EscM	Reverse index. Move the cursor up one line, scrolling if necessary.
Esc	Set default state.
	Set all variables to startup state.
	Set the video mode to a text mode with 80 columns and the current rows setting.
	Set tabs to every 8 columns
	Set to text color
	Set scroll area to the entire screen
	Clear the screen
	Set to nowrap and replace mode
	Set bell and cursor to defaults

Terminal Mode Keyboard

When the keyboard is set to terminal mode the following sequences are returned when non-character keys are pressed. These codes are based on the AIX values.

Note: The Esc[q values are returned with leading zeros for AIX compatibility. Note that ANSI parsing states that Esc[1q and Esc[001q are equivalent.

Function Keys:

F1	Esc[001q
F2	Esc[002q
F3	Esc[003q
F4	Esc[004q
F5	Esc[005q
F6	Esc[006q
F7	Esc[007q
F8	Esc[008q
F9	Esc[009q
F10	Esc[010q
F11	Esc[011q
F12	Esc[012q
Shift-F1	Esc[013q
Shift-F2	Esc[014q
Shift-F3	Esc[015q
Shift-F4	Esc[016q
Shift-F5	Esc[017q
Shift-F6	Esc[018q

Shift-F7	Esc[019q
Shift-F8	Esc[020q
Shift-F9	Esc[021q
Shift-F10	Esc[022q
Shift-F11	Esc[023q
Shift-F12	Esc[024q
Ctrl-F1	Esc[025q
Ctrl-F2	Esc[026q
Ctrl-F3	Esc[027q
Ctrl-F4	Esc[028q
Ctrl-F5	Esc[029q
Ctrl-F6	Esc[030q
Ctrl-F7	Esc[031q
Ctrl-F8	Esc[032q
Ctrl-F9	Esc[032q
Ctrl-F10	Esc[034q
Ctrl-F11	Esc[035q
Ctrl-F12	Esc[036q
Alt-F1	Esc[037q
Alt-F2	Esc[038q
Alt-F3	Esc[039q
Alt-F4	Esc[040q
Alt-F5	Esc[041q
Alt-F6	Esc[042q
Alt-F7	Esc[043q
Alt-F8	Esc[044q
Alt-F9	Esc[045q
Alt-F10	Esc[046q
Alt-F11	Esc[047q
Alt-F12	Esc[048q

Action Keys:

Backtab	Esc[Z
Del	Esc[P
Down	Esc[B
End	Esc[146q
Home	Esc[H
Ins	Esc[139q
Left	Esc[D

Pad5	Esc[121q
PgDn	Esc[154q
PgUp	Esc[150q
Right	Esc[C
Up	Esc[A

Alternate Keys:

Alt-0	Esc[067q
Alt-1	Esc[058q
Alt-2	Esc[059q
Alt-3	Esc[060q
Alt-4	Esc[061q
Alt-5	Esc[062q
Alt-6	Esc[063q
Alt-7	Esc[064q
Alt-8	Esc[065q
Alt-9	Esc[066q
Alt-A	Esc[087q
Alt-B	Esc[105q
Alt-C	Esc[103q
Alt-D	Esc[089q
Alt-E	Esc[076q
Alt-F	Esc[090q
Alt-G	Esc[091q
Alt-H	Esc[092q
Alt-I	Esc[081q
Alt-J	Esc[093q
Alt-K	Esc[094q
Alt-L	Esc[095q
Alt-M	Esc[107q
Alt-N	Esc[106q
Alt-O	Esc[082q
Alt-P	Esc[083q
Alt-Q	Esc[074q
Alt-R	Esc[077q
Alt-S	Esc[088q
Alt-T	Esc[078q
Alt-U	Esc[080q

Alt-V	Esc[104q
Alt-W	Esc[075q
Alt-X	Esc[102q
Alt-Y	Esc[079q
Alt-Z	Esc[101q
Alt-quotesingle	Esc[099q
Alt-comma	Esc[109q
Alt-hyphen	Esc[068q
Alt-period	Esc[111q
Alt-slash	Esc[113q
Alt-grave	Esc[115q
Alt-semicolon	Esc[097q
Alt-equals	Esc[070q
Alt-bracketleft	Esc[084q
Alt-backslash	Esc[086q
Alt-bracketright	Esc[085q
Alt-Backspace	Esc[071q
Alt-Del	Esc[M
Alt-Down	Esc[166q
Alt-End	Esc[149q
Alt-Enter	Esc[100q
Alt-Enter	Esc[100q
Alt-ErEof	Esc[119q
Alt-Esc	Esc[122q
Alt-Home	Esc[145q
Alt-Ins	Esc[141q
Alt-Left	Esc[160q
Alt-Padstar	Esc[188q
Alt-Padplus	Esc[201q
Alt-Padminus	Esc[199q
Alt-Padslash	Esc[180q
Alt-PgDn	Esc[157q
Alt-PgUp	Esc[153q
Alt-Right	Esc[169q
Alt-Tab	Esc[073q
Alt-Up	Esc[163q

Control Keys:

Ctrl-0	Esc[056q
--------	----------

Ctrl-1	Esc[049q
Ctrl-2	Esc[135q
Ctrl-3	Esc[050q
Ctrl-4	Esc[051q
Ctrl-5	Esc[052q
Ctrl-6	Esc[136q
Ctrl-7	Esc[053q
Ctrl-8	Esc[054q
Ctrl-9	Esc[055q
Ctrl-hyphen	Esc[057q
Ctrl-equals	Esc[069q
Ctrl-Del	Esc[142q
Ctrl-Down	Esc[165q
Ctrl-End	Esc[148q
Ctrl-Home	Esc[144q
Ctrl-Ins	Esc[140q
Ctrl-Left	Esc[159q
Ctrl-Padstar	Esc[187q
Ctrl-Padplus	Esc[198q
Ctrl-Padminus	Esc[198q
Ctrl-Padslash	Esc[182q
Ctrl-Pad5	Esc[184q
Ctrl-PgDn	Esc[156q
Ctrl-PgUp	Esc[152q
Ctrl-PrtSc	Esc[211q
Ctrl-Right	Esc[168q
Ctrl-Tab	Esc[072q
Ctrl-Up	Esc[162q

Generic IOCtl Commands

The following is a list of IOCtl commands that are currently supported on OS/2 Warp (PowerPC Edition). For a complete description of all OS/2 Warp Version 3 IOCtl commands see *OS/2 Warp Version 3 OS/2 Control Program Programming Reference* , Appendix F.

01h	Serial Device Control
	All functions are supported except:
54h	Set Enhanced Mode Parameters
74h	Query Enhanced Mode Parameters
03h	Video Control

None of the Video Control functions are supported.

04h

Keyboard Control

All functions are supported except:

50h	Set Code Page
52h	Set Interim Character Flags
57h	Set KCB
59h	Set Read/Pek Notification
5Ah	Alter Keyboard LEDs
5Ch	Set NLS and Custom Code Page
5Dh	Create a Logical Keyboard
5Eh	Destroy a Logical Keyboard
72h	Query Interim Character Flags

The following functions are partially supported:

53h	High-order byte not supported
73h	High-order byte not supported
79h	High-order byte of ShiftKeyStat not supported.
	KbdDD Flags not supported
	Xlate Flags not supported
	Xlate State 1 not supported
	Xlate State 2 not supported.

05h

Parallel Port Control

All functions are supported except:

48h	Activate Font
51h	Set Multiple Hardware Access
69h	Query Active Font
6Ah	Verify Font

The following 05h functions are emulated in the LD library instead of the device driver.

42h	Set Frame Control
44h	Set Infinite Retry
62h	Query Frame Control
64h	Query Infinite Retry

07h

Mouse Control

All functions are supported except:

55h	Reassign Mouse Threshold Values
5Ah	Set OS/2-Mode Pointer Draw Device Driver Address
5Dh	Notification of Mode Switch Completion
69h	Query Mouse Threshold Values

08h

Logical Disk Control

All functions are supported except:

03h	Set Logical Map
21h	Query Logical Map
40h	Removable Media Control
5Dh	Diskette Control
60h	Query Media Sense

The following functions are partially supported:

45h	Support only for floppy drives with complete tracks and standard formats (720 Kb, 1.2 Mb, 1.44 Mb, and 2.88 Mb).
66h	Support for Bit 2 only.

09h

Physical Disk Control

None of the Physical Disk Control functions are supported.

0Ah

Character Device Monitor Control

None of the Character Device Monitor Control functions are supported.

0Bh

General Device Control

All functions are supported except:

41h	System Notification
60h	Query Monitor Support

0Ch

Advanced Power Management

	None of the Advanced Power Management functions are supported.
80h	Screen Control None of the Screen Control functions are supported.
80h	OEMHLP Controls None of the OEMHLP Control functions are supported.
80h	Adapter Presence Check Control None of the Adapter Presence functions are supported.
80h	CD-ROM Drive and Disk Control All functions are supported except:
	62h Read Bytes
81h	CD-ROM Audio Control All functions are supported.
81h	Touch-Device-Dependent Driver Control None of the Touch-Device-Dependent functions are supported.
81h	Touch-Device-Independent Driver Control None of the Touch-Device-Independent functions are supported.

Metric Data Pool Definition

This section describes the data provided for metric tools.

All NON_UMA standard metrics start with a class id of 254, and IBM_Microkernel standard metrics start with a class id of 254.1.

The following components are currently provided:

NON_UMA	254
IBM_MICROKERNEL	254.1 ...
GLOBAL_HOST	254.1.1 ...
GENERAL_INFO	254.1.1.1
INFO	254.1.1.2
TASK	254.1.2 ...
GENERAL_INFO	254.1.2.1
BASIC_INFO	254.1.2.2
THREAD	254.1.3 ...
GENERAL_INFO	254.1.3.1
BASIC_INFO	254.1.3.2
SCHED_INFO	254.1.3.3
PROCESSOR	254.1.4 ...
SET_INFO	254.1.4.1
BASIC_INFO	254.1.4.2
SCHEDULER	254.1.5 ...
RUN_INFO	254.1.5.1
LPT_INFO	254.1.5.2

HOST_INFO	254.1.5.3
TASK_INFO	254.1.5.4
MEMORY	254.1.6 ...
GENERAL_INFO	254.1.6.1
DP_MEM_INFO	254.1.6.2
VM_REGION_INFO	254.1.6.3
DP_MEM_REQUESTS	254.1.6.4

Global Host General Information (254.1.1)

Global information for the host for displays which can handle uchar data and want the cpu model id.

Class ID: **254.1.1.1**

DCI Instance ID Information:

Number of levels	1
Number of instances	1

DCI Instance Level structure for Level 1, ID **HostCntlInst**:

type	UMA_UINT32
itype	DCI_HOSTPORT
Instance id size	4 bytes

Output data structure: [host_basic_info](#)

host_basic_info

Defines basic information about a host.

```
typedef struct _host_basic_info {
    UMA_UINT16      TOPOLOGY;          /* The processor topology of the system. */
    UMA_UINT16      FIRMWARE_FLAG;     /* TRUE if the system is PREP compliant. */
    UMA_UINT32      MAX_CPUS;          /* The total number of CPUs on the host. */
    UMA_UINT32      AVAIL_CPUS;        /* The total number of available CPUs. */
    UMA_OCTET_STRING_48 MODEL_ID_NUM;  /* The model id of the system. */
    UMA_UINT32      GLOBAL_MEMORY_SIZE; /* The size of memory in bytes. */
} host_basic_info;

typedef host_basic_info *host_basic_info;
```

host_basic_info Field - TOPOLOGY

TOPOLOGY (UMA_UINT16)
The processor topology of the system.

Datum ID	1
Units	DCI_UNITS_INFO DCI_CPU
Flags	DCI_QUERYABLE

host_basic_info Field - FIRMWARE_FLAG

FIRMWARE_FLAG (UMA_UINT16)
TRUE if the system is PREP compliant.

Datum ID	2
Units	DCI_UNITS_INFO DCI_TRUEFALSE
Flags	DCI_QUERYABLE

host_basic_info Field - MAX_CPUS

MAX_CPUS (UMA_UINT32)
The total number of CPUs on the host.

Datum ID	3
Units	DCI_UNITS_INFO DCI_CPU
Flags	DCI_QUERYABLE

host_basic_info Field - AVAIL_CPUS

AVAIL_CPUS (UMA_UINT32)
The total number of available CPUs.

Datum ID	4
Units	DCI_UNITS_INFO DCI_CPU
Flags	DCI_QUERYABLE

host_basic_info Field - MODEL_ID_NUM

MODEL_ID_NUM (UMA_OCTET_STRING_48)
The model id of the system.

Datum ID	5
Units	DCI_UNITS_INFO DCI_MODEL_ID
Flags	DCI_QUERYABLE

host_basic_info Field - GLOBAL_MEMORY_SIZE

GLOBAL_MEMORY_SIZE (UMA_UINT32)
The size of memory in bytes.

Datum ID	6
Units	DCI_UNITS_INFO DCI_MEMORY
Flags	DCI_QUERYABLE

Global Host Information (254.1.2)

Global information for the host.

Class ID: **254.1.1.2**

DCI Instance ID Information:

Number of levels	1
Number of instances	1

DCI Instance Level structure for Level 1, ID **HostCntInst**:

type	UMA_UINT32
itype	DCI_HOSTPORT
Instance id size	4 bytes

Output data structure: [ps_host_basic_info](#)

ps_host_basic_info

Defines basic information about a host.

```
typedef struct _ps_host_basic_info {  
    UMA_UINT16    TOPOLOGY;          /* The processor topology of the system. */  
    UMA_UINT16    FIRMWARE_FLAG;     /* TRUE if the system is PREP compliant. */  
    UMA_UINT32    MAX_CPUS;           /* The total number of CPUs on the host. */  
    UMA_UINT32    AVAIL_CPUS;         /* The total number of available CPUs. */  
    UMA_UINT32    GLOBAL_MEMORY_SIZE; /* The size of memory in bytes. */  
} ps_host_basic_info;  
  
typedef ps_host_basic_info *ps_host_basic_info;
```


ps_host_basic_info Field - TOPOLOGY

TOPOLOGY (UMA_UINT16)
The processor topology of the system.

Datum ID	1
Units	DCI_UNITS_INFO DCI_CPU
Flags	DCI_QUERYABLE

ps_host_basic_info Field - FIRMWARE_FLAG

FIRMWARE_FLAG (UMA_UINT16)
TRUE if the system is PREP compliant.

Datum ID	2
Units	DCI_UNITS_INFO DCI_TRUEFALSE
Flags	DCI_QUERYABLE

ps_host_basic_info Field - MAX_CPUS

MAX_CPUS (UMA_UINT32)
The total number of CPUs on the host.

Datum ID	3
Units	DCI_UNITS_INFO DCI_CPU
Flags	DCI_QUERYABLE

ps_host_basic_info Field - AVAIL_CPUS

AVAIL_CPUS (UMA_UINT32)
The total number of available CPUs.

Datum ID	4
Units	DCI_UNITS_INFO DCI_CPU
Flags	DCI_QUERYABLE

ps_host_basic_info Field - GLOBAL_MEMORY_SIZE

GLOBAL_MEMORY_SIZE (UMA_UINT32)
The size of memory in bytes.

Datum ID	5
Units	DCI_UNITS_INFO DCI_MEMORY
Flags	DCI_QUERYABLE

Task General Information (254.1.2.1)

Detailed task information.

Class ID: **254.1.2.1**

DCI Instance ID Information:

Number of levels	1
Number of instances	As many instances as there are tasks on the system.

DCI Instance Level structure for Level 1, ID **TaskInstId**:

type	UMA_UINT32
itype	DCI_TASKPORT
Instance id size	4 bytes

Output data structure: [task_data_ps](#)

task_data_ps

Defines detailed task information.

```
typedef struct _task_data_ps {
    UMA_UINT32      THREAD_COUNT_CURRENT; /* Count of current threads. */
    UMA_TIMESPEC    USER_TIME_LIVE;      /* Total user run time for live threads within the task. */
    UMA_TIMESPEC    SYSTEM_TIME_LIVE;    /* Total system run time for live threads within the task. */
    UMA_UINT32      SUSPEND_COUNT;       /* The current suspend count for the task. */
    UMA_UINT32      BASE_PRIORITY;       /* Base scheduling priority for the task. */
    UMA_UINT32      VIRTUAL_SIZE;        /* The number of virtual pages for the task. */
    UMA_UINT32      RESIDENT_SIZE;       /* The number of resident pages for the task. */
    UMA_TIMESPEC    USER_TIME_TERM;      /* The total user run time for terminated threads within the task. */
    UMA_UINT32      SYSTEM_TIME_TERM;    /* The total system run time for terminated threads within the task. */
    UMA_UINT32      TASK_KERNELID_ADDR;  /* Task kernel id address used for trace and performance. */
} task_data_ps;

typedef task_data_ps *task_data_ps;
```

task_data_ps Field - THREAD_COUNT_CURRENT

THREAD_COUNT_CURRENT (UMA_UINT32)
Count of current threads.

Datum ID	1
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

task_data_ps Field - USER_TIME_LIVE

USER_TIME_LIVE (UMA_TIMESPEC)
Total user run time for live threads within the task.

Datum ID	2
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

task_data_ps Field - SYSTEM_TIME_LIVE

SYSTEM_TIME_LIVE (UMA_TIMESPEC)
Total system run time for live threads within the task.

Datum ID	3
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

task_data_ps Field - SUSPEND_COUNT

SUSPEND_COUNT (UMA_UINT32)
The current suspend count for the task.

Datum ID	4
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

task_data_ps Field - BASE_PRIORITY

BASE_PRIORITY (UMA_UINT32)
Base scheduling priority for the task.

Datum ID	5
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

task_data_ps Field - VIRTUAL_SIZE

VIRTUAL_SIZE (UMA_UINT32)
The number of virtual pages for the task.

Datum ID	6
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

task_data_ps Field - RESIDENT_SIZE

RESIDENT_SIZE (UMA_UINT32)
The number of resident pages for the task.

Datum ID	7
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

task_data_ps Field - USER_TIME_TERM

USER_TIME_TERM (UMA_TIMESPEC)
The total user run time for terminated threads within the task.

Datum ID	8
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

task_data_ps Field - SYSTEM_TIME_TERM

SYSTEM_TIME_TERM (UMA_UINT32)

The total system run time for terminated threads within the task.

Datum ID	9
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

task_data_ps Field - TASK_KERNELID_ADDR

TASK_KERNELID_ADDR (UMA_UINT32)

Task kernel id address used for trace and performance.

Datum ID	10
Units	DCI_UNITS_INFO DCI_ADDR
Flags	DCI_QUERYABLE

Task Basic Information (254.1.2.2)

Task basic information for all tasks on the system.

Class ID: **254.1.2.2**

DCI Instance ID Information:

Number of levels	1
Number of instances	As many instances as there are tasks on the system.

DCI Instance Level structure for Level 1, ID **TaskInstId**:

type	UMA_UINT32
itype	DCI_TASKPORT
Instance id size	4 bytes

Output data structure: [ps_task_basic_info](#)

ps_task_basic_info

Defines basic information about a task.

```
typedef struct _ps_task_basic_info {  
    UMA_UINT32    THREAD_COUNT_CURRENT; /* Count of current threads. */  
    UMA_UINT32    SUSPEND_COUNT;        /* Current suspend count for the task. */  
    UMA_UINT32    BASE_PRIORITY;        /* Base scheduling priority for the task. */  
    UMA_UINT32    VIRTUAL_SIZE;         /* Number of virtual pages for the task. */  
    UMA_UINT32    RESIDENT_SIZE;        /* Number of resident pages for the task. */  
    UMA_UINT32    USER_TIME_TERM;      /* Total user run time for terminated threads within the task. */  
}
```

```

UMA_UINT32      SYSTEM_TIME_TERM;      /* Total system run time for terminated threads within the task. */
UMA_UINT32      TASK_KERNEL_ID_ADDR;    /* Address of the task kernel ID used for trace and performance. */
} ps_task_basic_info;

typedef ps_task_basic_info *ps_task_basic_info;

```

ps_task_basic_info Field - THREAD_COUNT_CURRENT

THREAD_COUNT_CURRENT (UMA_UINT32)
Count of current threads.

Datum ID	1
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

ps_task_basic_info Field - SUSPEND_COUNT

SUSPEND_COUNT (UMA_UINT32)
Current suspend count for the task.

Datum ID	2
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

ps_task_basic_info Field - BASE_PRIORITY

BASE_PRIORITY (UMA_UINT32)
Base scheduling priority for the task.

Datum ID	3
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

ps_task_basic_info Field - VIRTUAL_SIZE

VIRTUAL_SIZE (UMA_UINT32)
Number of virtual pages for the task.

Datum ID	4
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

ps_task_basic_info Field - RESIDENT_SIZE

RESIDENT_SIZE (UMA_UINT32)
Number of resident pages for the task.

Datum ID	5
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

ps_task_basic_info Field - USER_TIME_TERM

USER_TIME_TERM (UMA_UINT32)
Total user run time for terminated threads within the task.

Datum ID	6
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

ps_task_basic_info Field - SYSTEM_TIME_TERM

SYSTEM_TIME_TERM (UMA_UINT32)
Total system run time for terminated threads within the task.

Datum ID	7
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

ps_task_basic_info Field - TASK_KERNEL_ID_ADDR

TASK_KERNEL_ID_ADDR (UMA_UINT32)
Address of the task kernel ID used for trace and performance.

Datum ID	8
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

Thread General Information (254.1.3.1)

Detailed task information.

Class ID: **254.1.2.1**

DCI Instance ID Information:

Number of levels	2
Number of instances	As many instances as there are threads on the system.

DCI Instance Level structure for Level 1, ID **TaskInstId**:

type	UMA_UINT32
itype	DCI_TASKPORT
Instance id size	4 bytes

DCI Instance Level 2 structure for Level 2, ID **ThreadInstId**:

type	UMA_UINT32
itype	DCI_THREADPORT
Instance id size	4 bytes

Output data structure: [task_thread_data_ps](#)

task_thread_data_ps

Defines general information about a thread.

```
typedef struct _task_thread_data_ps {
    UMA_TIME_SPEC    USER_TIME;          /* Total thread user time. */
    UMA_UINT32       FLAGS;              /* Thread flags. */
    UMA_UINT32       SUSPEND_COUNT;      /* Thread suspend count. */
    UMA_UINT32       SLEEP_TIME;         /* Thread sleep time. */
    UMA_UINT32       THREAD_KERNEL_ID_ADDR; /* Address of the thread kernel ID used for trace and performance. */
    UMA_UINT32       POLICY;             /* Scheduling policy in effect. */
    UMA_UINT32       DATA;              /* Associated data for the scheduling policy. */
    UMA_UINT32       BASE_PRIORITY;      /* Base scheduling priority. */
    UMA_UINT32       MAX_PRIORITY;       /* Maximum scheduling priority. */
    UMA_UINT32       CUR_PRIORITY;       /* Current scheduling priority. */
    UMA_UINT32       DEPRESSED;          /* TRUE if scheduling priority is depressed. */
    UMA_UINT32       DEPRESS_PRIORITY;   /* Scheduling priority from depressed. */
} task_thread_data_ps;

typedef task_thread_data_ps *task_thread_data_ps;
```

task_thread_data_ps Field - USER_TIME

USER_TIME (UMA_TIME_SPEC)
Total thread user time.

Datum ID	1
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

task_thread_data_ps Field - FLAGS

FLAGS (UMA_UINT32)
Thread flags.

Datum ID	7
Units	DCI_UNITS_INFO DCI_THREADID
Flags	DCI_QUERYABLE

task_thread_data_ps Field - SUSPEND_COUNT

SUSPEND_COUNT (UMA_UINT32)
Thread suspend count.

Datum ID	8
Units	DCI_UNITS_INFO DCI_COUNT
Flags	DCI_QUERYABLE

task_thread_data_ps Field - SLEEP_TIME

SLEEP_TIME (UMA_UINT32)
Thread sleep time.

Datum ID	9
Units	DCI_UNITS_TIME DCI_DCI_MILISECS
Flags	DCI_QUERYABLE

task_thread_data_ps Field - THREAD_KERNEL_ID_ADDR

THREAD_KERNEL_ID_ADDR (UMA_UINT32)

Address of the thread kernel ID used for trace and performance.

Datum ID	10
Units	DCI_UNITS_INFO DCI_ADDR
Flags	DCI_QUERYABLE

task_thread_data_ps Field - POLICY

POLICY (UMA_UINT32)

Scheduling policy in effect.

Datum ID	11
Units	DCI_UNITS_INFO DCI_ORDER
Flags	DCI_QUERYABLE

task_thread_data_ps Field - DATA

DATA (UMA_UINT32)

Associated data for the scheduling policy.

Datum ID	12
Units	DCI_UNITS_INFO DCI_DATA
Flags	DCI_QUERYABLE

task_thread_data_ps Field - BASE_PRIORITY

BASE_PRIORITY (UMA_UINT32)

Base scheduling priority.

Datum ID	13
Units	DCI_UNITS_INFO DCI_PRECEDENCE
Flags	DCI_QUERYABLE

task_thread_data_ps Field - MAX_PRIORITY

MAX_PRIORITY (UMA_UINT32)
Maximum scheduling priority.

Datum ID	14
Units	DCI_UNITS_INFO DCI_PRECEDENCE
Flags	DCI_QUERYABLE

task_thread_data_ps Field - CUR_PRIORITY

CUR_PRIORITY (UMA_UINT32)
Current scheduling priority.

Datum ID	15
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

task_thread_data_ps Field - DEPRESSED

DEPRESSED (UMA_UINT32)
TRUE if scheduling priority is depressed.

Datum ID	16
Units	DCI_UNITS_INFO DCI_TRUEFALSE
Flags	DCI_QUERYABLE

task_thread_data_ps Field - DEPRESS_PRIORITY

DEPRESS_PRIORITY (UMA_UINT32)
Scheduling priority from depressed.

Datum ID	17
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

Thread Basic Information (254.1.3.2)

Detailed Thread information.

Class ID: **254.1.3.2**

DCI Instance ID Information:

Number of levels	2
Number of instances	As many instances as there are threads on the system.

DCI Instance Level structure for Level 1, ID **TaskInstId**:

type	UMA_UINT32
itype	DCI_TASKPORT
Instance id size	4 bytes

DCI Instance Level structure for Level 2, ID **ThreadInstId**:

type	UMA_UINT32
itype	DCI_THREADPORT
Instance id size	4 bytes

Output data structure: [thread_basic_info](#)

thread_basic_info

Detailed thread information.

```
typedef struct _thread_basic_info {
    UMA_TIME_SPEC    USER_TIME;           /* Total thread user time. */
    UMA_TIME_SPEC    SYSTEM_TIME;         /* Total thread system time. */
    UMA_UINT32       CPU_USAGE;            /* Total thread CPU usage. */
    UMA_UINT32       BASE_PRIORITY;        /* Thread basic priority. */
    UMA_UINT32       CUR_PRIORITY;         /* Thread current priority. */
    UMA_UINT32       RUN_STATE;            /* Thread run state. */
    UMA_UINT32       FLAGS;                /* Thread flags. */
    UMA_UINT32       SUSPEND_COUNT;        /* Thread suspend count. */
    UMA_UINT32       SLEEP_TIME;           /* Thread sleep time. */
    UMA_UINT32       THREAD_KERNEL_ID_ADDR; /* Address of the thread kernel ID used for trace and performance. */
} thread_basic_info;

typedef thread_basic_info *thread_basic_info;
```

thread_basic_info Field - USER_TIME

USER_TIME (UMA_TIME_SPEC)
Total thread user time.

Datum ID	1
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

thread_basic_info Field - SYSTEM_TIME

SYSTEM_TIME (UMA_TIME_SPEC)
Total thread system time.

Datum ID	2
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

thread_basic_info Field - CPU_USAGE

CPU_USAGE (UMA_UINT32)
Total thread CPU usage.

Datum ID	3
Units	DCI_UNITS_TIME DCI_MILISECS
Flags	DCI_QUERYABLE

thread_basic_info Field - BASE_PRIORITY

BASE_PRIORITY (UMA_UINT32)
Thread basic priority.

Datum ID	4
Units	DCI_UNITS_INFO DCI_PRECEDENCE
Flags	DCI_QUERYABLE

thread_basic_info Field - CUR_PRIORITY

CUR_PRIORITY (UMA_UINT32)
Thread current priority.

Datum ID	5
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

thread_basic_info Field - RUN_STATE

RUN_STATE (UMA_UINT32)
Thread run state.

Datum ID	6
Units	DCI_UNITS_INFO DCI_THREADID
Flags	DCI_QUERYABLE

thread_basic_info Field - FLAGS

FLAGS (UMA_UINT32)
Thread flags.

Datum ID	7
Units	DCI_UNITS_INFO DCI_THREADID
Flags	DCI_QUERYABLE

thread_basic_info Field - SUSPEND_COUNT

SUSPEND_COUNT (UMA_UINT32)
Thread suspend count.

Datum ID	1
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

thread_basic_info Field - SLEEP_TIME

SLEEP_TIME (UMA_UINT32)
Thread sleep time.

Datum ID	9
Units	DCI_UNITS_TIME DCI_MILISECS
Flags	DCI_QUERYABLE

thread_basic_info Field - THREAD_KERNEL_ID_ADDR

THREAD_KERNEL_ID_ADDR (UMA_UINT32)

Address of the thread kernel ID used for trace and performance.

Datum ID	10
Units	DCI_UNITS_INFO DCI_ADDR
Flags	DCI_QUERYABLE

Thread Scheduler Information (254.1.3.3)

Thread scheduling information.

Class ID: **254.1.3.3**

DCI Instance ID Information:

Number of levels	2
Number of instances	As many instances as there are threads on the system.

DCI Instance Level structure for Level 1, ID **TaskInstId**:

type	UMA_UINT32
itype	DCI_TASKPORT
Instance id size	4 bytes

DCI Instance Level structure for Level 2, ID **ThreadInstId**:

type	UMA_UINT32
itype	DCI_THREADPORT
Instance id size	4 bytes

Output data structure: [thread_sched_info](#)

thread_sched_info

Thread scheduling information.

```
typedef struct _thread_sched_info {
    UMA_UINT32    POLICY;          /* Scheduling policy in effect. */
    UMA_UINT32    DATA;          /* Associated data for the scheduling policy. */
    UMA_UINT32    BASE_PRIORITY;  /* Base scheduling priority. */
    UMA_UINT32    MAX_PRIORITY;   /* Maximum scheduling priority. */
    UMA_UINT32    CUR_PRIORITY;   /* Current scheduling priority. */
    UMA_UINT32    DEPRESSED;      /* TRUE if scheduling priority is depressed. */
    UMA_UINT32    DEPRESS_PRIORITY; /* Scheduling priority from depressed. */
} thread_sched_info;

typedef thread_sched_info *thread_sched_info;
```

thread_sched_info Field - POLICY

POLICY (UMA_UINT32)
Scheduling policy in effect.

Datum ID	1
Units	DCI_UNITS_INFO DCI_ORDER
Flags	DCI_QUERYABLE

thread_sched_info Field - DATA

DATA (UMA_UINT32)
Associated data for the scheduling policy.

Datum ID	2
Units	DCI_UNITS_INFO DCI_DATA
Flags	DCI_QUERYABLE

thread_sched_info Field - BASE_PRIORITY

BASE_PRIORITY (UMA_UINT32)
Base scheduling priority.

Datum ID	3
Units	DCI_UNITS_INFO DCI_PRECEDENCE
Flags	DCI_QUERYABLE

thread_sched_info Field - MAX_PRIORITY

MAX_PRIORITY (UMA_UINT32)
Maximum scheduling priority.

Datum ID	4
Units	DCI_UNITS_INFO DCI_PRECEDENCE
Flags	DCI_QUERYABLE

thread_sched_info Field - CUR_PRIORITY

CUR_PRIORITY (UMA_UINT32)
Current scheduling priority.

Datum ID	5
Units	DCI_UNITS_INFO DCI_COUNT
Flags	DCI_QUERYABLE

thread_sched_info Field - DEPRESSED

DEPRESSED (UMA_UINT32)
TRUE if scheduling priority is depressed.

Datum ID	6
Units	DCI_UNITS_INFO DCI_TRUEFALSE
Flags	DCI_QUERYABLE

thread_sched_info Field - DEPRESS_PRIORITY

DEPRESS_PRIORITY (UMA_UINT32)
Scheduling priority from depressed.

Datum ID	7
Units	DCI_UNITS_INFO DCI_COUNT
Flags	DCI_QUERYABLE

Processor Set Information (254.1.4.1)

Processor set information for a processor set on the host.

Class ID: **254.1.4.1**

DCI Instance ID Information:

Number of levels	1
Number of instances	1

DCI Instance Level structure for Level 1, ID **ProcCtrlInst**:

type	UMA_UINT32
itype	DCI_PROCSCTLPORT
Instance id size	4 bytes

Output data structure: [processor_set_load_info](#)

processor_set_load_info

Processor set information.

```
typedef struct _processor_set_load_info {  
    UMA_UINT32    TASK_COUNT;    /* Total tasks currently in this processor set. */  
    UMA_UINT32    THREAD_COUNT;  /* Total threads currently in this processor set. */  
    UMA_UINT32    LOAD_AVERAGE; /* Load average per CPU. */  
    UMA_UINT32    MACH_FACTOR;   /* Processor resources available to a new thread. */  
} processor_set_load_info;  
  
typedef processor_set_load_info *processor_set_load_info;
```

processor_set_load_info Field - TASK_COUNT

TASK_COUNT (UMA_UINT32)
Total tasks currently in this processor set.

Datum ID	1
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

processor_set_load_info Field - THREAD_COUNT

THREAD_COUNT (UMA_UINT32)
Total threads currently in this processor set.

Datum ID	2
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

processor_set_load_info Field - LOAD_AVERAGE

LOAD_AVERAGE (UMA_UINT32)
Load average per CPU.

Average number of runnable processors divided by the number of CPUs, scaled by LOAD_SCALE.

Datum ID	3
----------	---

Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

processor_set_load_info Field - MACH_FACTOR

MACH_FACTOR (UMA_UINT32)

Processor resources available to a new thread.

Datum ID	4
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

Processor Basic Information (254.1.4.2)

Processor basic information for a processor on the host.

Class ID: **254.1.4.2**

DCI Instance ID Information:

Number of levels	1
Number of instances	1

DCI Instance Level structure for Level 1, ID **ProcCtrlInst**:

type	UMA_UINT32
itype	DCI_PROCSCTLPORT
Instance id size	4 bytes

Output data structure: [processor_set_load_info](#)

processor_basic_info

Processor basic information.

```
typedef struct _processor_basic_info {
    UMA_UINT32    CPU_TYPE;           /* Type of CPU. */
    UMA_UINT32    CPU_SUBTYPE;        /* Sub-type of CPU. */
    UMA_UINT32    SLOT_NUMBER;        /* Slot number of CPU. */
    UMA_UINT32    PROCESSOR_HZ;       /* Processor speed. */
    UMA_UINT32    LOCAL_MEMORY_SIZE;  /* Local memory size. */
    UMA_UINT32    RUNNING;            /* TRUE if CPU is running. */
    UMA_UINT32    IS_MASTER;          /* TRUE if this is the master processor. */
} processor_basic_info;

typedef processor_basic_info *processor_basic_info;
```

processor_basic_info Field - CPU_TYPE

CPU_TYPE (UMA_UINT32)
Type of CPU.

Datum ID	1
Units	DCI_UNITS_INFO DCI_CPU
Flags	DCI_QUERYABLE

processor_basic_info Field - CPU_SUBTYPE

CPU_SUBTYPE (UMA_UINT32)
Sub-type of CPU.

Datum ID	2
Units	DCI_UNITS_INFO DCI_MODEL
Flags	DCI_QUERYABLE

processor_basic_info Field - SLOT_NUMBER

SLOT_NUMBER (UMA_UINT32)
Slot number of CPU.

Datum ID	3
Units	DCI_UNITS_INFO DCI_POSITION
Flags	DCI_QUERYABLE

processor_basic_info Field - PROCESSOR_HZ

PROCESSOR_HZ (UMA_UINT32)
Processor speed.

Datum ID	4
Units	DCI_UNITS_INFO DCI_PROCESSOR_SPEED
Flags	DCI_QUERYABLE

processor_basic_info Field - LOCAL_MEMORY_SIZE

LOCAL_MEMORY_SIZE (UMA_UINT32)
Local memory size.

Datum ID	5
Units	DCI_UNITS_INFO DCI_MEMORY
Flags	DCI_QUERYABLE

processor_basic_info Field - RUNNING

RUNNING (UMA_UINT32)
TRUE if CPU is running.

Datum ID	6
Units	DCI_UNITS_INFO DCI_TRUEFALSE
Flags	DCI_QUERYABLE

processor_basic_info Field - IS_MASTER

IS_MASTER (UMA_UINT32)
TRUE if this is the master processor.

Datum ID	7
Units	DCI_UNITS_INFO DCI_TRUEFALSE
Flags	DCI_QUERYABLE

Processor Basic Information (254.1.5.1)

The average number of threads in a runnable state.

Class ID: **254.1.5.1**

DCI Instance ID Information:

Number of levels	1
Number of instances	1

DCI Instance Level structure for Level 1, ID **HostCntInst**:

type UMA_UINT32
itype DCI_HOSTPORT
Instance id size 4 bytes

Output data structure: [host_load_info](#)

host_load_info

Scheduler run information.

```
typedef struct _host_load_info {  
    UMA_UINT32    AVENRUN_1;    /* Average run over 5 seconds. */  
    UMA_UINT32    AVENRUN_2;    /* Average run over 30 seconds. */  
    UMA_UINT32    AVENRUN_3;    /* Average run over 60 seconds. */  
    UMA_UINT32    MACH_FACTOR_1; /* Processing resources available to a new thread. */  
    UMA_UINT32    MACH_FACTOR_2; /* Processing resources available to a new thread. */  
    UMA_UINT32    MACH_FACTOR_3; /* Processing resources available to a new thread. */  
} host_load_info;  
  
typedef host_load_info *host_load_info;
```

host_load_info Field - AVENRUN_1

AVENRUN_1 (UMA_UINT32)

Average run over 5 seconds.

The average number of runnable processes divided by the number of CPUs over three periods of time: 5 seconds.

Datum ID	1
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

host_load_info Field - AVENRUN_2

AVENRUN_2 (UMA_UINT32)

Average run over 30 seconds.

The average number of runnable processes divided by the number of CPUs over three periods of time: 30 seconds.

Datum ID	2
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

host_load_info Field - AVENRUN_3

AVENRUN_3 (UMA_UINT32)

Average run over 60 seconds.

The average number of runnable processes divided by the number of CPUs over three periods of time: 60 seconds.

Datum ID	3
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

host_load_info Field - MACH_FACTOR_1

MACH_FACTOR_1 (UMA_UINT32)

Processing resources available to a new thread.

Datum ID	4
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

host_load_info Field - MACH_FACTOR_2

MACH_FACTOR_2 (UMA_UINT32)

Processing resources available to a new thread.

Datum ID	5
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

host_load_info Field - MACH_FACTOR_3

MACH_FACTOR_3 (UMA_UINT32)

Processing resources available to a new thread.

Datum ID	3
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

Scheduler Host Information (254.1.5.2)

Host scheduler information.

Class ID: **254.1.5.2**

DCI Instance ID Information:

Number of levels	1
Number of instances	1

DCI Instance Level structure for Level 1, ID **HostCntlInst**:

type	UMA_UINT32
itype	DCI_HOSTPORT
Instance id size	4 bytes

Output data structure: [host_sched_info](#)

host_sched_info

Host scheduler information.

```
typedef struct _host_sched_info {
    UMA_UINT32    MIN_TIMEOUT; /* Minimum timeout in milliseconds */
    UMA_UINT32    MIN_QUANTUM; /* Minimum quantum in milliseconds. */
} host_sched_info;

typedef host_sched_info *host_sched_info;
```

host_sched_info Field - MIN_TIMEOUT

MIN_TIMEOUT (UMA_UINT32)
Minimum timeout in milliseconds

Datum ID	1
Units	DCI_UNITS_INFO DCI_SECSMILLI
Flags	DCI_QUERYABLE

host_sched_info Field - MIN_QUANTUM

MIN_QUANTUM (UMA_UINT32)
Minimum quantum in milliseconds.

Datum ID	2
Units	DCI_UNITS_INFO DCI_SECSMILLI
Flags	DCI_QUERYABLE

Scheduler LPT Information (254.1.5.3)

Processor idle time.

Class ID: **254.1.5.3**

DCI Instance ID Information:

Number of levels	1
Number of instances	1

DCI Instance Level structure for Level 1, ID **HostCtrlInst**:

type	UMA_UINT32
itype	DCI_PRCCTLPORT
Instance id size	4 bytes

Output data structure: [processor_timeinfo_t](#)

processor_timeinfo_t

Processor idle time.

```
typedef struct _processor_timeinfo_t {
    UMA_TIMESPEC    IDLE_TIME;
    UMA_TIMESPEC    RESERVED1;
    UMA_TIMESPEC    RESERVED2;
} processor_timeinfo_t;

typedef processor_timeinfo_t *processor_timeinfo_t;
```

processor_timeinfo_t Field - IDLE_TIME

IDLE_TIME (UMA_TIMESPEC)

Total time since system boot that this processor was idle.

Datum ID	1
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

processor_timeinfo_t Field - RESERVED1

RESERVED1 (UMA_TIMESPEC)

Total time this processor was in supervisor time. (Reserved for future use.)

Datum ID	2
Units	DCI_UNITS_TIMESPEC DCI_NANOSECS
Flags	DCI_QUERYABLE

processor_timeinfo_t Field - RESERVED2

RESERVED2 (UMA_TIMESPEC)

Total time this processor was in user time. (Reserved for future use.)

Datum ID	3
Units	DCI_UNITS_TIMESPEC DCI_NANOSECS
Flags	DCI_QUERYABLE

Scheduler Task Information (254.1.5.4)

Returns live task scheduler information for all the tasks on the system.

Class ID: **254.1.5.4**

DCI Instance ID Information:

Number of levels	1
Number of instances	As many instances as there are tasks on the system.

DCI Instance Level structure for Level 1, ID **TaskInstId**:

type	UMA_UINT32
itype	DCI_TASKPORT
Instance id size	4 bytes

Output data structure: [task_thread_times_info](#)

task_thread_times_info

Live task scheduler information.

```
typedef struct _task_thread_times_info {
    UMA_TIMESPEC    USER_TIME;    /* Total user time for live threads. */
    UMA_TIMESPEC    SYSTEM_TIME;  /* Total system time for live threads. */
} task_thread_times_info;

typedef task_thread_times_info *task_thread_times_info;
```

task_thread_times_info Field - USER_TIME

USER_TIME (UMA_TIMESPEC)
Total user time for live threads.

Datum ID	1
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

task_thread_times_info Field - SYSTEM_TIME

SYSTEM_TIME (UMA_TIMESPEC)
Total system time for live threads.

Datum ID	2
Units	DCI_UNITS_TIME DCI_NANOSECS
Flags	DCI_QUERYABLE

Memory General Information (254.1.6.1)

Virtual memory statistics of the microkernel.

Class ID: **254.1.6.1**

DCI Instance ID Information:

Number of levels	1
Number of instances	As many instances as there are tasks on the system.

DCI Instance Level structure for Level 1, ID **HostCntlInst**:

type	UMA_UINT32
itype	DCI_HOSTPORT
Instance id size	4 bytes

Output data structure: [task_data_ps](#)

vm_statistics

Virtual memory statistics of the microkernel.

```
typedef struct _vm_statistics {  
    UMA_UINT32    FREE_COUNT;        /* Total number of free pages in the system. */  
    UMA_UINT32    ACTIVE_COUNT;      /* Total number of pages currently in use and pageable. */  
    UMA_UINT32    INACTIVE_COUNT;    /* Number of inactive pages. */  
    UMA_UINT32    WIRE_COUNT;        /* Number of pages that are wired in memory and cannot be paged out. */  
    UMA_UINT32    ZERO_FILLED_COUNT; /* Number of zero-filled pages. */  
    UMA_UINT32    REACTIVATIONS;     /* Number of reactivated pages. */  
    UMA_UINT32    PAGEINS;           /* Number of requests for pages from a pager. */  
}
```

```

UMA_UINT32    PAGEOUTS;          /* Number of pages that have been paged out. */
UMA_UINT32    FAULTS;           /* Number of times the vm_fault routine has been called. */
UMA_UINT32    COW_FAULTS;       /* Number of copy-on write faults. */
UMA_UINT32    LOOKUPS;          /* Number of object cache lookups. */
UMA_UINT32    HITS;             /* Number of object cache hits. */
} vm_statistics;

typedef vm_statistics *vm_statistics;

```

vm_statistics Field - FREE_COUNT

FREE_COUNT (UMA_UINT32)
Total number of free pages in the system.

Datum ID	1
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - ACTIVE_COUNT

ACTIVE_COUNT (UMA_UINT32)
Total number of pages currently in use and pageable.

Datum ID	2
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - INACTIVE_COUNT

INACTIVE_COUNT (UMA_UINT32)
Number of inactive pages.

Datum ID	3
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - WIRE_COUNT

WIRE_COUNT (UMA_UINT32)

Number of pages that are wired in memory and cannot be paged out.

Datum ID	4
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - ZERO_FILLED_COUNT

ZERO_FILLED_COUNT (UMA_UINT32)

Number of zero-filled pages.

Datum ID	5
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - REACTIVATIONS

REACTIVATIONS (UMA_UINT32)

Number of reactivated pages.

Datum ID	6
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - PAGEINS

PAGEINS (UMA_UINT32)

Number of requests for pages from a pager.

Datum ID	7
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - PAGEOUTS

PAGEOUTS (UMA_UINT32)

Number of pages that have been paged out.

Datum ID	8
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - FAULTS

FAULTS (UMA_UINT32)

Number of times the vm_fault routine has been called.

Datum ID	9
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - COW_FAULTS

COW_FAULTS (UMA_UINT32)

Number of copy-on write faults.

Datum ID	10
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - LOOKUPS

LOOKUPS (UMA_UINT32)

Number of object cache lookups.

Datum ID	11
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

vm_statistics Field - HITS

HITS (UMA_UINT32)
Number of object cache hits.

Datum ID	12
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

Memory DP Mem Information (254.1.6.2)

Paging statistics for the default pager.

Class ID: **254.1.6.2**

DCI Instance ID Information:

Number of levels	1
Number of instances	As many instances as there are tasks on the system.

DCI Instance Level structure for Level 1, ID **DefPgrInst**:

type	UMA_UINT32
itype	DCI_DPGRPORT
Instance id size	4 bytes

Output data structure: [default_pager_info](#)

default_pager_info

Paging statistics for the default pager.

```
typedef struct _default_pager_info {  
    UMA_UINT32    DPI_TOTAL_SPACE;  
    UMA_UINT32    DPI_FREE_SPACE;  
    UMA_UINT32    DPI_PAGE_SIZE;  
} default_pager_info;  
  
typedef default_pager_info *default_pager_info;
```

default_pager_info Field - DPI_TOTAL_SPACE

DPI_TOTAL_SPACE (UMA_UINT32)
Total space in bytes in the default manager's currently allocated paging partitions.

Datum ID	1
Units	DCI_UNITS_INFO DCI_BYTES
Flags	DCI_QUERYABLE

default_pager_info Field - DPI_FREE_SPACE

DPI_FREE_SPACE (UMA_UINT32)

Free space in bytes in the default manager's currently allocated paging partitions.

Datum ID	2
Units	DCI_UNITS_INFO DCI_BYTES
Flags	DCI_QUERYABLE

default_pager_info Field - DPI_PAGE_SIZE

DPI_PAGE_SIZE (UMA_UINT32)

Default manager's VM page size.

Datum ID	3
Units	DCI_UNITS_INFO DCI_BYTES
Flags	DCI_QUERYABLE

Memory VM Region Information (254.1.6.3)

Obtains the current attributes for all vm regions in all of the tasks on the system.

Class ID: **254.1.6.3**

DCI Instance ID Information:

Number of levels	2
Number of instances	One for each memory region for every task on the system.

DCI Instance Level structure for Level 1, ID **TaskInstId**:

type	UMA_UINT32
itype	DCI_TASKPORT
Instance id size	4 bytes

DCI Instance Level structure for Level 2, ID **MRegAddr**:

type	UMA_UINT32
itype	DCI_VMADDRESS
Instance id size	4 bytes

Output data structure: [vm_region_info](#)

vm_region_info

Current attributes for vm regions.


```

typedef struct _vm_region_info {
    UMA_UINT32    START_ADDRESS;    /* Address of the region. */
    UMA_UINT32    SIZE_OF_REGION;    /* Size of the region. */
    UMA_UINT32    PROTECTION;        /* Current protection for the region. */
    UMA_UINT32    MAX_PROTECTION;    /* Maximum protection allowed for the region. */
    UMA_UINT32    INHERITANCE;       /* Inheritance attribute for the region. */
    UMA_UINT32    SHARED;            /* If TRUE, the region is shared by another task. */
    UMA_UINT32    OBJECT_NAME;       /* Name of the object. */
    UMA_UINT32    OFFSET;            /* Region's offset into the memory object. */
} vm_region_info;

typedef vm_region_info *vm_region_info;

```

vm_region_info Field - START_ADDRESS

START_ADDRESS (UMA_UINT32)
Address of the region.

Datum ID	1
Units	DCI_UNITS_INFO DCI_ADDR
Flags	DCI_QUERYABLE

vm_region_info Field - SIZE_OF_REGION

SIZE_OF_REGION (UMA_UINT32)
Size of the region.

Datum ID	2
Units	DCI_UNITS_INFO DCI_COUNT
Flags	DCI_QUERYABLE

vm_region_info Field - PROTECTION

PROTECTION (UMA_UINT32)
Current protection for the region.

Datum ID	3
Units	DCI_UNITS_INFO DCI_PROTECT
Flags	DCI_QUERYABLE

vm_region_info Field - MAX_PROTECTION

MAX_PROTECTION (UMA_UINT32)
Maximum protection allowed for the region.

Datum ID	4
Units	DCI_UNITS_INFO DCI_PROTECT
Flags	DCI_QUERYABLE

vm_region_info Field - INHERITANCE

INHERITANCE (UMA_UINT32)
Inheritance attribute for the region.

Datum ID	5
Units	DCI_UNITS_INFO DCI_TRUEFALSE
Flags	DCI_QUERYABLE

vm_region_info Field - SHARED

SHARED (UMA_UINT32)
If TRUE, the region is shared by another task.

Datum ID	6
Units	DCI_UNITS_INFO DCI_MEMORY
Flags	DCI_QUERYABLE

vm_region_info Field - OBJECT_NAME

OBJECT_NAME (UMA_UINT32)
Name of the object.

Datum ID	7
Units	DCI_UNITS_INFO DCI_OBJECTNAME
Flags	DCI_QUERYABLE

vm_region_info Field - OFFSET

OFFSET (UMA_UINT32)
Region's offset into the memory object.

Datum ID	8
Units	DCI_UNITS_INFO DCI_OFFSET
Flags	DCI_QUERYABLE

Memory DP Mem Requests (254.1.6.4)

Returns a set of statistics on the default pager.

Class ID: **254.1.6.4**

DCI Instance ID Information:

Number of levels	1
Number of instances	One for each paging space on the system.

DCI Instance Level structure for Level 1, ID **PagSpaceInst**:

type	UMA_UINT32
itype	DCI_DPGRPORT
Instance id size	4 bytes

Output data structure: [pager_info_ps](#) form=textonly.

pager_info_ps

Statistics on the default pager.

```
typedef struct _pager_info_ps {
    UMA_UINT32    PAGE_IN_REQUESTS; /* Number of page-in requests. */
    UMA_UINT32    PAGE_OUT_REQUESTS; /* Number of page-out requests. */
    UMA_UINT32    STATE; /* State of the paging space. */
    UMA_UINT32    TOTAL_PAGES;
    UMA_UINT32    FREE_PAGE;
    UMA_UINT32    HIGH_WATER_MARK; /* High water mark for the paging space. */
    UMA_UINT32    LOW_WATER_MARK; /* Low water mark for the paging space. */
} pager_info_ps;

typedef pager_info_ps *pager_info_ps;
```

pager_info_ps Field - PAGE_IN_REQUESTS

PAGE_IN_REQUESTS (UMA_UINT32)
Number of page-in requests.

Datum ID	1
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

pager_info_ps Field - PAGE_OUT_REQUESTS

PAGE_OUT_REQUESTS (UMA_UINT32)
Number of page-out requests.

Datum ID	2
Units	DCI_UNITS_COUNT DCI_COUNT
Flags	DCI_QUERYABLE

pager_info_ps Field - STATE

STATE (UMA_UINT32)
State of the paging space.

Datum ID	3
Units	DCI_UNITS_INFO DCI_STATE
Flags	DCI_QUERYABLE

pager_info_ps Field - TOTAL_PAGES

TOTAL_PAGES (UMA_UINT32)
The total number of pages in the paging space that can be used for page_in and page_out. This field is valid only if the state is appropriate.

Datum ID	4
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

pager_info_ps Field - FREE_PAGE

FREE_PAGE (UMA_UINT32)
The number of pages in the paging space that are not backing any memory object. These pages can be used for page_in and page_out. This field is valid only if the state is appropriate.

Datum ID	5
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

pager_info_ps Field - HIGH_WATER_MARK

HIGH_WATER_MARK (UMA_UINT32)
High water mark for the paging space.

This field is valid only if the state of the paging space indicates that it can be used for both page_in and page_out.

Datum ID	6
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

pager_info_ps Field - LOW_WATER_MARK

LOW_WATER_MARK (UMA_UINT32)
Low water mark for the paging space.

This field is valid only if the state of the paging space indicates that it can be used for both page_in and page_out.

Datum ID	7
Units	DCI_UNITS_COUNT DCI_PAGES
Flags	DCI_QUERYABLE

Notices

First Edition (January 1996)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

(C) Copyright International Business Machines Corporation 1994, 1995. All rights reserved.

[Performance Services](#) of this book incorporates text which is copyright 1994 X/Open Company Limited. The text was taken by permission from *X/Open UMA DCI Preliminary Specification (Draft December 1994)*, ISBN: 1-85912-068-7.

Note to U.S. Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department RM1A, 1000 N.W. 51st Street, Boca Raton, FL 33431, U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
Common User Access
CUA
AT
FFST
First Failure Support Technology
IBM
OS/2
PowerPC
Presentation Manager
SAA
Systems Application Architecture
WIN-OS/2
Workplace Shell
XGA

The following terms are trademarks of other companies:

Bell
C++
CORBA
DEC
Intel
LaserJet
MASM
Microsoft

AT&T Bell Laboratories
AT&T, Inc.
Object Management Group, Inc.
Digital Equipment Corporation
Intel Corporation
Hewlett-Packard Company
Microsoft Corporation.
Microsoft Corporation.

Open Software Foundation
OpenDoc
OpenGL
OSF
PCMCIA
Postscript
Times New Roman
Times New Roman Bold Italic
VT102
Windows
X/Open
X-Windows

Open Software Foundation.
Apple Corporation.
Silicon Graphics Inc.
Open Software Foundation.
Personal Computer Memory Card International Association.
Adobe Systems Incorporated
Monotype Corporation, Limited
Monotype Corporation, Limited
Digital Equipment Corporation
Microsoft Corporation
X/Open Company Limited
Massachusetts Institute of Technology

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Glossary

This glossary includes terms and definitions from:

- The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994. Copies may be purchased from McGraw-Hill or in bookstores.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol (A) after the definition.
- The *ANSI/EIA Standard-440A: Fiber Optic Terminology*, copyright 1989 by the Electronics Industries Association (EIA). Copies can be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue N.W., Washington, D.C. 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Glossary Listing

Select a starting letter of glossary terms:

A	N
B	O
C	P
D	Q
E	R
F	S
G	T
H	U
I	V
J	W
K	X
L	Y
M	Z

Glossary - A

ABI Application binary interface.

access control list (ACL) A list associated with an object that identifies who can access the object and how the object can be accessed (for example read-only or write-only access).

ACL Access control list.

active window The window that a user is currently interacting with. The window that is active has a highlighted border and title bar. Contrast with *inactive window*.

Advanced Interactive Executive (AIX) IBM's implementation of the UNIX** operating system.

AIX Advanced Interactive Executive.

alias An alternate label, or name; for example, a name and one or more aliases may be used to refer to the same file. (A)

alternate personality Any operating mode, or personality, in an environment that supports multiple operating modes, other than the dominant personality. An alternate personality does not control the desktop on display screens. The MVM Personality is an example of an alternate personality.

American National Standard Code for Information Interchange (ASCII) The standard code, using a coded character set consisting of 7-bit coded characters (8-bit including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

American National Standards Institute (ANSI) An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. (A)

ANSI American National Standards Institute.

API Application programming interface.

APM Advanced Power Management.

application (1) The use to which an information processing system is put, for example, a payroll application, an airline reservation application, or a network application. (2) A collection of software components used to perform specific types of user-oriented work on a computer.

application binary interface (ABI) Linkage conventions for a particular microprocessor. (These conventions define, how registers are used, how they are saved and destroyed across calls, how parameters are passed, and so on.)

application program (1) A program that is specific to the solution of an application problem. Synonymous with "application software." (T) (2) A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. (3) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application programming interface (API) A functional interface supplied by the operating system or by a separately licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

array (1) An arrangement of data in one or more dimensions: a list, a table, or a multidimensional arrangement of items. (2) In programming languages, an aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting. (I)

ASCII American National Standard Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8-bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

ASCIIZ format A string of ASCII characters ending with a null character.

assignment A unique setting that can be changed. See also *button assignment* and *gesture assignment*.

asymmetric multiprocessing Unequal distribution of tasks across multiple processors.

asynchronous I/O A series of input/output operations that are being done separately from the job that requested them.

attribute A named property of an entity.

authentication server Part of the trusted security base, a server responsible for authenticating identities of clients. Maintains passwords and group membership information for users.

Glossary - B

backlight The fluorescent lighting on a liquid crystal display (LCD).

BIDI Bidirectional bus.

bidirectional bus (BIDI) A bus on which data can be sent in either direction.

big endian See *endian*.

bit map A representation of an image by an array of bits.

bit map resource A custom bit map that an application uses when it is displayed or as an item in a menu.

boot To prepare a computer system for operation by loading an operating system.

bootstrap (1) A sequence of instructions whose execution causes additional instructions to be loaded and executed until the complete computer program is in storage. (T) (2) A technique or device designed to bring itself into a desired state by means of its own action; for example, a machine routine whose first few instructions are sufficient to bring the rest of itself into the computer from an input device. (A) (3) A small program that loads larger programs during system initialization. (4) To execute a bootstrap. The term bootstrapping is also used for translating a compiler by using the compiler or a previous version as the translator. (T) See also *bootstrap loader*.

bootstrap loader An input routine in which preset computer operations are used to load a bootstrap. (I) (A) See also *bootstrap*.

buffer (1) A routine or storage used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another. (A) (2) A portion of storage used to hold input or output data temporarily.

button A mechanism used to request or initiate an action. See also *barrel buttons* and *bezel buttons*.

button assignment An assignment that defines the action or actions performed as a result of activating a button or combination of buttons.

byte (1) A string that consists of a number of bits, treated as a unit, and representing a single byte character. (T) (2) A binary character operated upon as a unit and usually shorter than a computer word. (A) (3) A group of 8 adjacent binary digits that represent one EBCDIC character. (4) See n-bit byte. See double byte character set.

Glossary - C

cache (1) A special-purpose buffer storage, smaller and faster than main storage, used to hold a copy of instructions and data obtained from main storage and likely to be needed next by the processor. (T) (2) A buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

calibration The adjustment of a piece of equipment so that it meets normal operational standards. For example, calibration could refer to adjusting the alignment of a pen

call (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) To transfer control to a procedure, program, routine, or subroutine.

Card Services A software management interface in the PCMCIA architecture that allows users to allocate the system resources (such as memory and interrupts) automatically, as soon as the Socket Services detects that a PC card has been added. Card Services also releases these resources when the PC Card has been removed.

cathode ray tube display (CRT display) (1) A device that presents data in visual form by means of controlled electron beams. (A) (2) The data display produced by such a device.

cdll Circular doubly linked list.

child class See *subclass*.

child process A process, started by a parent process, that shares the resources of the parent process.

child window A window that appears within the border of its parent window (either a primary window or another child window.) When the parent window is resized, moved, or destroyed, the child window is also resized, moved or destroyed. However, the child window can be moved or resized independently from the parent window, within the boundaries of the parent window. Contrast with *parent window*.

CID See *Configuration, Installation, and Distribution*.

class The description of a set of objects and their behavior. New classes can be defined in terms of existing classes through a technique

known as *inheritance* .

class method An action that can be performed on a class object. Class methods are also called factory methods.

class object The run-time implementation of a class.

click To press and release a mouse button without moving the pointer off the choice. See *double-click*.

client In a Workplace architecture client/server environment, the consumer of a service. An example is an application or shared service using a client library to communicate with the Workplace Naming Services to retrieve information from the system's name space.

client area The part of the window, inside the border, that is below the menu bar. It is the user's work space, where a user types information and selects choices from selection fields. In primary windows, it is where an application programmer presents the objects that a user works on.

client credentials The set of data associated with a client such as user identifier, group identifiers, roles (such as administrator), and special permissions.

Client Library A collection of executable personality neutral code and data that is bound to an application and provides the API of a Workplace shared service to clients. The functions of the service API may be implemented in the library or the library may map them to requests to a server or microkernel service.

client/server model In the Workplace architecture, the Mach-defined environment in which a small group of services (servers) at the system layer support a large group of clients (users) at the application layer via interprocess communication.

client window The window in which the application displays output and receives input. This window is located inside the frame window, under the window title bar and any menu bar, and within any scroll bars.

clipboard An area of computer memory or storage that temporarily holds data. Data in the clipboard is available to other applications.

clustered multiprocessing Distribution of tasks across sets of multiple processors.

coalesce (1) To combine two or more sets of items into one set of any form. (I) (2) To combine two or more files into one file.

composite window A window composed of other windows (such as a frame window, frame-control windows, and a client window) that are kept together as a unit and that interact with each other.

compound document A document that contains linked or embedded information created by other applications.

configure To describe to a system the devices, optional features, and programs installed on the system.

Configuration, Installation, and Distribution (CID) An IBM architecture used to automate installation and customization for Workplace and other products. CID enables LAN-connected machines to be installed and maintained remotely.

control ball In computer graphics, a ball, rotatable about its center, that is used as an input device, normally as a locator. (I) (A)
Synonymous with track ball.

copy To place onto the clipboard a duplicate of what has been selected.

CORBA Common Object Request Broker Architecture. A standard for distributed Group (OMG); object management.

core shared service A shared service that is included with OS/2 for PowerPC or Workplace products and can always be counted on to be present. These include name services, file services, pipe services, print/spooler services, loader services, internationalization services, event and windows services, LAN transport services, installation services, and software serviceability services.

CRT display Cathode ray tube display.

customize To make a personal version of something, for example, a voice model.

Glossary - D

database A collection of data with a given structure of accepting, storing, and providing, on demand, data for multiple users. (T)

data object The means by which a client and server exchange information that is represented by an array of descriptors.

DBCS See *double byte character set* .

DBCS Enabling DBCS enabling adds the capability to a product for the management of the double byte character set. All data and components in the system that handle a human readable character string use DBCS enabling.

DCB Device control block.

DCE Distributed computing environment.

DDE Dynamic data exchange.

DDK Device driver kit.

desktop A folder that fills the entire screen and holds all the objects with which the user can interact to perform operations on the system.

device control block (DCB) A data structure that contains data and addresses for indirect calls for a logical device.

device driver A program that enables the computer to communicate with a specific peripheral device, such as a pen, video disc player, or CD drive.

device driver kit A set of programming tools and device driver source code provided for device-driver developers.

device ID A unique identification number assigned to a logical device. A device ID is unique across an entire system.

device object An object that provides a means of communication between a computer and another piece of equipment, such as a pen or display.

DFS Distributed file system.

dialog (1) The interaction between a user and a computer. (2) In an interactive system, a series of related inquiries and responses similar to a conversation between two people.

dialog box A movable window, fixed in size, containing controls that a user uses to provide information required by an application so that it can continue to process a user request.

dialog template A series of statements that define the identifier, memory options, dialog-box dimensions, and controls of a dialog.

digital signal processor. (DSP) A high-speed coprocessor designed to do real-time manipulation of signals.

digitizer An electronic device that transmits coordinate data to software running on a computer.

directive A statement that carries out a task such as including a header file, defining constants, or conditionally compiling portions of a resource script file.

directory A type of file containing the names and controlling information for other files or other directories.

Distributed Computing Environment (DCE) A set of standards governed by the Open Software Foundation (OSF) for open computing in distributed systems.

Distributed file system A file system composed of files or directories that physically reside on more than one computer in a communication network.

Distributed management system IBM corporate framework for system management.

Distributed presentation management In IMS/VS, a message format service (MFS) option that allows programs to communicate with device independence by sharing message formatting functions between MFS and a user-written remote program. The user-written remote program performs device-dependent formatting.

Distributed Presentation Services (DPS) In DPPX systems, an IBM licensed program that provides the COBOL and assembler language application programmer with support for formatting and modifying the layout of data on displays and printers.

Distributed SOM (DSOM) Distributed System Object Model. Provides remote access to SOM objects in a transparent way that insulates client programmers from having to know the location or platform type where a target object will be instantiated. DSOM allows programmers to use the same object model independently of whether the objects they access are in the same process, in another process on the same machine, or across distributed networks.

DLL Dynamic link library.

DMS *Distributed management system* .

domain (1) In the OS/2 operating system, a collection of network objects that a group of users have authorization to use. (2) In speech-recognition systems, a set of vocabularies and language models that are designed to support a particular speech application.

domain ID Domain identification. A short name (or identifier) that is uniquely associated with a domain.

dominant personality A personality that is started first, "provides" the desktop (most likely, the desktop is an application of the dominant personality), and exports a set of support functions to the alternate personalities and Workplace Shared Services.

DOS Disk Operating System, the original operating system used with IBM-compatible personal computers.

double-byte character set (DBCS) A character set whose character codes are represented by two-byte code. Used for code pages for some Asian countries.

double-click To press and release a mouse button twice within a time frame defined by the user, without moving the pointer off the choice.
See *click*

double-tap To touch a touch-sensitive screen with a pointing device twice in rapid succession within a small area.

DPM Distributed presentation management.

DPS Distributed Presentation Services (DPS).

drag To use a pointing device to move an object.

drop To fix the position of an object that is being dragged by releasing the select button on the pointing device.

DSOM Distributed System Object Model.

DSP Digital signal processor.

dynamic data exchange (DDE) In Workplace architecture, a messaging protocol that allows PM applications to exchange data via shared memory. DDE transactions always take place between a client application and a server application. The client initiates a dialog with the server by requesting data from the server. The server responds by providing the requested data to the client.

dynamic link library (DLL) A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously. DLLs can be interchanged without recompiling or relinking the applications that use them.

Glossary - E

EA Extended attribute.

ELF Executable and linkable format.

EMMI External memory manager interface (of the IBM Microkernel product).

EMS expanded memory specification.

emulation The use of one system to imitate another system so that the imitating system accepts the same data, runs the same programs, and achieves the same results as the imitated system. Emulation is usually achieved by means of hardware or firmware. (T)

endian The addressing model that determines the byte ordering of both data and instructions that are stored in computer memory. The big endian model assigns the lowest address to the highest-order or most significant byte of a multibyte scalar data item. The little endian model assigns the lowest address to the lowest-order or least significant byte of a multibyte scalar data item.

event and window services A Workplace core shared service that provides the mechanism for sharing the console device among personalities and applications. It also provides services for the management of events and windows.

executable and linkable format (ELF) In the Workplace architecture, the object module format.

expanded memory specification (EMS) A memory specification that enables DOS applications to access memory above the 1MB real mode addressing limit.

extended attribute (EA) Additional information, such as comments, history, or author's name, that a system or program associates with a file. The system or program then uses this information to recognize the file.

extended memory specification Programming interface for DOS programs to access memory above 1MB.

Glossary - F

FAT File allocation table.

FEA Full extended attribute.

FEVSH File, Edit, View, Selected, Help menu system

file allocation table (FAT) In IBM personal computers, a table used by DOS to allocate space on a disk for a file and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in random or sequential manner. In contrast with High Performance File System (HPFS) which will perform faster on a large disk with less fragmentation.

File, Edit, View, Selected, Help menu system (FEVSH) Selectable menu items found at the top of a window.

File Server operations (FS ops) Shared Services file-system API's.

File Services A Workplace core shared service that provides a file system framework for multiple logical and physical file systems.

file system In Workplace architecture, software that supports storing data on a storage device. File systems manage the physical locations of data on the storage devices for applications. File systems also manage I/O operations and control the format of the stored data.

firm word During dictation, a word that has been recognized and will not be changed by the *word-usage model* when subsequent words are recognized. Contrast with *infirm word*.

First Failure Data Capture (FFDC) A methodology for decreasing the need for the reproduction of user failures by capturing the data associated with a failure as soon as it is detected. The FFDC methodology includes defensive programming, code instrumentation, event tracing, and logging facilities supported by FFST tools.

First Failure Support Technology (FFST) A set of functions that applications can use for problem determination. FFST functions include logging and displaying errors and messages, formatting and routing generic alerts, and generating data dumps.

flag A characteristic of a file or directory that enables it to be used in certain ways.

font resource A bit map that defines the shape of the individual characters in a character set.

frame-control window A control window, owned by the frame window, that accepts simple input from the user and notifies the frame window or client window when it receives input.

frame window The window used as a base when constructing a primary window or composite window. The frame window coordinates the action of frame controls and the client window, enabling them to act as a single unit.

framework A software package used to provide application programmers with a consistent, easy-to-use set of services. A framework exports an API for a set of functions that can be provided by multiple software vendors.

framework service provider Code that performs the functions the applications associated with a set of service provider interfaces (SPIs). A framework exports an API to client applications. It translates to a service provider interface (SPI) that can be supported by multiple service providers. This translation enables applications to work with service providers from multiple vendors and frees programmers from the need to use a different set of interfaces each time a new product of a given class is introduced.

front panel A container that "follows" the user from work area to work area; holds objects and actions that need to be accessed across work areas (printer, shredder, Shut Down, Lockup).

FS ops File Server operations.

full extended attribute A data structure that contains an extended attribute's name and value. Used when querying, adding, deleting, and changing EAs.

fully qualified path A path specification that begins with either a drive specifier (x:) or a path separator character (\) preceded by zero, one, or two periods.

function (1) A specific purpose of an entity, or its characteristic action. (A) (2) A machine action such as a carriage return or a line feed. (A) (3) A subroutine that returns the value of a single variable and that usually has a single exit; for example, subroutines that compute mathematical functions, such as sine, cosine, logarithm, or that compute the maximum of a set of numbers. (T)

Glossary - G

GEA Get extended attribute.

gesture A hand-drawn symbol or uppercase letter which, when recognized by the system, invokes an action or a series of actions. Gestures denote an action and a target for the action.

gesture assignment An assignment that defines the action or actions performed as a result of a gesture.

get extended attribute A data structure that contains an extended attribute's name. Used when querying extended attributes.

get user attribute (GUA) A data structure that contains a user attribute. Used when querying user attributes.

GPI Graphics programming interface.

graphical user interface (GUI) In SAA Common User Access architecture, a type of user interface that takes advantage of high-resolution graphics. In common usage, a graphical user interface includes a combination of graphics, the object-action paradigm, the use of pointing devices, menu bars and other menus, overlapping windows, and icons.

graphics engine The drawing engine for the system. It manages display resources, such as color maps, bit maps, and serves as the interface between the graphics transportation protocols and the presentation (display) and printer device drivers.

graphics programming interface (GPI) Programming interface defined for service provider response to system requests for presentation device services. The mechanism by which applications, controls, and the desktop access the graphics engine in order to display characters on a screen or printer.

graphics transportation protocols The method by which graphics data is communicated between a client (application) and a server (graphics engine).

GUA Get user attribute.

GUI Graphical user interface

Glossary - H

Hardware Resource Manager A component of the IBM Microkernel that provides services for configuration, access, and management of hardware to user level device drivers in the Microkernel Services run-time environment.

hashing (1) A method of transforming a search key into an address for the purpose of storing and retrieving items of data. The method is often designed to minimize the search time. (T) (2) The application of an algorithm to the records in a data set to obtain a symmetric grouping of the records. Synonymous with key folding. (3) In an indexed dataset, the process of transforming a record key into an index value for storing and retrieving a record.

hash table A table of information that is accessed by way of a shortened search key (the hash value). Using a hash table minimizes average search time.

hash total (1) The result obtained by applying an algorithm to a set of heterogeneous data for checking purposes; for example, a summation obtained by treating data items as numbers. (T) (2) Synonym for control total.

header (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message such as one or more destination fields, name of originating station, input sequence number, character string indicating the type of message, and priority level for the message.

high-performance file system (HPFS) In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the coexistence of multiple, active file systems on a single personal computer, with the capability of multiple and different storage devices. File names used with the HPFS can have as many as 254 characters. On larger disks HPFS will out perform FAT with less fragmentation.

hot key (1) The key combination used to change from one session to another on the workstation. (2) To jump, or hot key, from a host session to an application on the workstation, or from the workstation to the host session.

hot spot The area of a display screen that is activated to accept user input. Synonymous with *touch* area.

HPFS High-performance file system.

HRM Hardware Resource Manager.

Glossary - I

IAC Interapplication communication.

IBM Microkernel Product The IBM implementation of microkernel technology. It includes the microkernel and a set of Microkernel Services that can be used to create operating system personality and other services that execute on the microkernel.

icon resource A bit map that defines the shape of the icon to be used for a given application.

IDL Interface definition language

IFS Installable file systems.

inactive window A window that a user is not currently interacting with.

Industry Standard Architecture (ISA) Bus architecture on Family 1 personal computers.

inheritance (1) In the OS/2 Presentation Manager interface, the technique of specifying the shape and behavior of one window (called a *child* window) as incremental differences from another window (called the *parent window*). (2) In System Object Model, implications of derivation of new classes from existing classes. Child classes inherit methods from parent and ancestor classes.

i-node The internal structure that describes the individual files in the operating system; there is one i-node for each file. An i-node contains the node, type, owner, and location of a file. A table of i-nodes is stored near the beginning of a file system. Synonymous with file index. See also vnode.

Input Method Editor (IME) IME is a program that generates a sequence of characters for applications. IME normally intercepts characters from an input stream and then returns converted characters back to the stream. IME usually generates characters not assigned on a keyboard or not recognized in other input, such as pen input. It may show the interim state of the conversion which often involves some user interface. That is why it is called an editor. Asian and Arabic countries require IME in order to support a large character set and the bidirectional input respectively.

install helper object A Workplace Shell object used to initialize install objects.

install object A Workplace Shell object that can be used to install software applications.

install object template A template used to create new installable software objects.

installation component database A file containing entries about directories, files, links, permissions (attributes to DOS), owner, and group, where the file is to be copied, and where the file is to be copied from.

Installation Frameworks A framework for aiding developers in building installation packages. The installation frameworks program provides a consistent user interface for software installations.

installation registry A description of products and services installed on a computer, including licenses owned by the user and the level of software on the system. The installation is stored in a registry subsystem.

Installation Services A core shared service that provides a common installation facility for OS/2 for PowerPC.

installation stanza description A control file on the startup installation diskette that describes the startup method (how to boot).

intelligent shadows Shadows (that remember specific-settings data) that are separate and apart from the object that they are shadowing.

interapplication communication Information transfer between applications, such as DDE and OLE.

internationalization (Internationalization is referred to by an abbreviation I18N because there are eighteen letters between the first I and the last N.) See *national language support*.

interprocess communication (IPC) In the Workplace architecture, the asynchronous messaging facility for cooperating subsystems to communicate through ports that provides the basis for the Workplace client-server system model. See also Message Interface Generator.

inventory object A Workplace Shell object that contains information about currently installed software.

invoke To start a command, procedure, or program. To call a subroutine by means of a CALL statement or by a defined assignment. To call a function by a reference to it during the evaluation of an expression.

IPC Interprocess communication

ISA Industry Standard Architecture.

Glossary - J

JFS Journal file system.

Journal file system (JFS) A special-purpose file that is used to record changes made in the system.

joy stick In computer graphics, a lever that can pivot in all directions and that is used as a locator device. (T)

Glossary - K

kernel The set of basic operating-system services that runs at the highest priority level. The part of an operating system that performs basic functions such as allocating hardware resources.

kernel security token Each client is tagged with a security token in kernel address space. The token is set only by trusted code and represents the user's credentials.

Glossary - L

LAN Local area network.

large media extended partition (LMEP) An architecture that supports large partitions (greater than one gigabyte) and is a superset of the current disk architecture.

lazy drag A mouse action in which a user can pick up a set of objects and carry those objects around on the pointer without having to hold the mouse button down.

LCD Liquid crystal display.

LED Light-emitting diode.

legacy code For operating systems, code developed for earlier environments that must be supported by any new environment.

lift-off point Location plotted by the digitizer where a pointing device is removed from a touch-sensitive surface.

linear executable format The object module format used by OS/2 on Intel** platforms.

little endian See *endian*.

LMEP Large media extended partition.

local area network (LAN) A computer network located on a user's premises within a limited geographical area. Communication within a LAN is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. Contrast with *wide area network* .

localization Localization is an architecture and/or design which a language and cultural differences are absorbed by the product. Localization is referred to by an abbreviation L10N because there are ten letters between the first L and the last N.

logical device (1) A file for conducting input or output with a physical device. (2)A file for mapping user I/O between virtual and real devices.

logical file system A component of Workplace file services that provides a consistent view of multiple physical file systems. The logical file system provides path resolution and other services that are independent of the on-disk format of data.

LX Linear executable format.

Glossary - M

Mach An operating system kernel developed at Carnegie-Mellon University to support distributed and parallel computation.

Machine-readable information (MRI) This usually refers to text information subject to international translation considerations.

menu resource A collection of information that defines the appearance and function of an application menu.

menu template A data structure used to define a menu.

message (1) A packet of data used for communication between the Presentation Manager interface and Presentation Manager applications.
(2) Information not requested by a user but displayed in response to an unexpected event, or when something undesirable could occur.

Message Interface Generator A Workplace tool that generates low-level remote procedure calls between a client and server process using the microkernel IPC mechanism. The code that is generated includes routines to pack and unpack the messages used to communicate between processes.

method An action that can be performed on an object.

Microkernel In the Workplace architecture, the component of the IBM Microkernel Product that runs in the most privileged state of the computer and controls the basic operation of the computer. It includes only those functions required to provide a set of abstract processing environments and to permit applications to work together as clients and servers. These are IPC, ports, tasks, threads, and virtual memory management.

microkernel services The system services provided with the IBM Microkernel product. There are three classes of services: Kernel Services, which control the basic operation of the machine; Microkernel Shared Services, which run on the microkernel and provide boot, default pager, and root name services; and Device Services, which include device and I/O management services.

MP Multiprocessor.

MRI Machine-readable information.

multiple virtual machines (MVM) personality The Workplace personality that manages the collection of Virtual x86[®] Machines. The MVM Personality provides DOS/WINDOWS/DPMI program compatibility on a PowerPC. It manages and supports execution of multiple x86 DOS and Windows programs (in real mode and protect mode) each in its own virtual x86 machine.

multiprocessor (1) A computer including two or more processor that have common access to a main storage. (2) A system of two or more processing units, or processors that can communicate without manual intervention.

multiuser Pertaining to two or more people who use the services of a processor within a given period of time; usage is normally serial unless otherwise specified.

MVM Multiple virtual machines.

MVM Server The server that manages the collection of Virtual x86 Machines.

Glossary - N

Name Services A core shared service that provides a programming interface for its clients to store and access transient or persistent systems and state information in the global name space. It exports APIs to its clients and defines SPIs for name service providers.

Name Services Entry Attribute Registry entry information, consisting of three descriptors: class identifier, attribute name, and attribute value.

Name Services mount A point or node in a client's registry name space tree established by a registry service provider and implemented as a registry subtree.

name space A hierarchical (tree) structure of nodes, with one distinguished node called the root. Each node is labeled by a distinct, nonempty string of characters called a component name. Nodes are directories or leaves (for example, terminal nodes that represent an entity identified by a mach port: file, device, root of a file set, and user name). Each node may have a list of attributes and access control. A name space is assigned naming rules during creation.

naming rules Rules that define how to create component names for a name space (including restricted characters, case sensitivity) and how to do path resolution for a name space.

National Language Feature (NLF) NLF is a group of components that support functions that relate to the national language and country unique requirements. An input method and font data are components of NLF.

national language support (NLS) The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product.

NLS National language support.

notification Session-related (asynchronous) status messages generated by the recognition engine and sent to subscribing applications.

Glossary - O

object (1) A graphic symbol that a user works with to perform a task. (2) In System Object Model, a set of data and the actions that can be performed on that data.

object definition (1) The set of information required to create and manage an object. (2) The creation of a control block for an object. It also defines the object as available to the user. See also *class*.

object handler Part of Event and Window Services, a handler that maintains window clipping regions, maps virtual color maps into one or more hardware color lookup tables, and controls updating of the current cursor shape on the window.

object instance An object belonging to a certain class.

object linking and embedding (OLE) An application protocol developed by Microsoft** Corp. that allows objects created by one application to be linked to or embedded in objects created by another application.

Object Management Group A common standards organization that works with Object Module Request Brokers to make sure there is consistency among Request Broker Vendors.

object module A binary executable or data component, or both, resulting from source assembly or compilation, or from secondary linkage of such object modules.

object module format A set of instructions in machine language produced by a compiler from a source program.

object name A construct that singles out a particular (directory) object from all other objects. The name must be unambiguous (that is, denote just one object). However, it need not be unique. For example, it may be the only name that unambiguously denotes the object).

object-oriented (1) Pertaining to a user interface in which a user works with objects rather than applications, to perform tasks. (2) Pertaining to programming, a method for structuring applications as hierarchically organized classes describing objects that may interact with other objects.

object package An XOM term that is similar in meaning to a service.

object subclass A class of objects that define new behavior in terms of new methods not inherited from parent and ancestor classes.

object value Arbitrarily complex information item that can be viewed as a characteristic or a property of an object.

OEM Original equipment manufacturer.

OLE Object linking and embedding

om attribute Component of an object, comprising an integer denoting the attribute's type and an ordered sequence of one or more attribute values, each accompanied with an integer denoting the value's syntax.

om attribute value Value in an om descriptor. Values are constrained according to the package that implements the object.

OMF Object module format.

OMG Object Management Group

OM object Workplace registry data object that consists of a name, some security data, a list of attributes, and a list of subentries (allowing for nesting).

OpenDoc A standard for compound documents and data exchange.

Open Software Foundation (OSF) A standards body for Unix and Microkernel to ensure consistency.

Open Systems Interconnection (OSI) (1)The interconnection of open systems in accordance with standards of the International Organization for Standardization (ISO) for the exchange of information. (T) (A) (2) The use of standardized procedures to enable the interconnection of data processing systems.

operating system (OS) Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output controls, and data management. Although operating systems are predominately software, partial hardware implementations are possible.(T)

option (1) A specification in a statement that may be used to influence the execution of the statement. (2) See default option.

option argument A command-line specification that is specified with one or more arguments to modify some default behavior.

original equipment manufacturer (OEM) A manufacturer of equipment that may be marketed by another manufacturer.

OS Operating system.

OS/2 operating system. A multitasking operating system with 32-bit architecture.

OS/2 Software Installer A set of tools for creating and packaging install objects.

OSF Open Software Foundation.

OSI Open Systems Interconnections.

Glossary - P

package object A group of related object classes.

page-in (1) In virtual storage systems, the process of transferring a page from external page storage to real storage. (2) The process of transferring a page from auxiliary storage to main storage.

page-out (1) In virtual storage systems, the process of transferring a page from real storage to external page storage. (2) The transfer of a page from main storage to auxiliary storage.

Palladium The open systems spooler.

parent class A class from which another class inherits. See also *inheritance*.

parent process A process that creates other processes. Contrast with *child process*.

parent window In the OS/2 operating system, a window that creates a child window. The child window is drawn within the parent window. If the parent window is moved, resized, or destroyed, the child window will also be moved, resized or destroyed. However, the child window can be moved and resized independently from the parent window, within the boundaries of the parent window.

path The part of a file specification that lists the drive name and a series of directory names that give the location of the file. Each directory name is separated by the backslash character. In the specification C:\MYFILES\MISC\GLOSSARY.SCR, the path consists of C:\MYFILES\MISC\.

path resolution The resolution of a path to the correct file services entity (directory or file).

PC Card A credit-card-sized adapter used to add memory, storage, or I/O capabilities to a personal computer, personal communicator, or other electronic device.

PCMCIA The Personal Computer Memory Card International Association.

PDA Personal Digital Assistant.

PDL Printer Description Language (such as PostScript**, PCL5, Dot Matrix).

pel See *picture element*.

persistence An attribute of an object that characterizes its lifespan. An object that remains in effect across time is said to be 'persistent'.

Personal Computer Memory Card International Association (PCMCIA) A nonprofit technical standards and trade association established to develop a common format and integrated-circuit PC Cards.

Personal Digital Assistant A small portable computer that acts on modules of code and can be used to store data such as phone numbers.

personality An operating-system-dependent application execution environment that corresponds to traditional operating systems such as OS/2, DOS, and AIX.

personality dependent (PD) Code that depends on APIs or services provided by a Workplace personality service.

personality neutral (PN) Code that is operating system independent because it depends only on Workplace Shared Services including the Microkernel.

phoneme A unit of sound distinguished by linguists and also found in pronunciation dictionaries. A phoneme has a variable duration of up to several seconds.

Physical File System. (PFS) A Workplace component that manages the on-disk storage, indexing, mounting, and recovery of data. The File Services framework supports the installation of multiple Physical File Systems. Examples of Physical File Systems include FAT, HPFS, and CDROM.

picture element In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (T)

pixel Picture element.

PM Presentation Manager.

PMATE Presentation Manager Automated Test Environment.

pointer (1) A data element that indicates the location of another data element. (T) (2) In computer graphics, a manually operated functional unit used to specify and addressable point. A pointer may be used to conduct interactive graphic operations, such as selection of one member of a predetermined set of display elements, or indication of a position on a display space while generating coordinate data. (T) (3) An identifier that indicates the location of an item of data. (A) (4) The symbol displayed on the screen that a user moves with a pointing device, such as a mouse. (5) A physical or symbolic identifier of a unique target. (6) In the C language, a variable that holds the address of a data object or a function.

pointer resource A bit map that defines the shape of the mouse pointer on the screen.

pointing device An instrument, such as a pen, mouse, trackball or joystick, used to move a pointer on the screen.

polyinstantiation The ability to allow multiple instances of a personality or application to run simultaneously.

pop-up window See *dialog box*.

port In the Workplace architecture, a unidirectional asynchronous communication channel between a client and a server. A port has a single receiver and may have multiple senders. If a reply is provided to a service request a second port must be used.

Portable Operating System Interfaces for Computer Environments An IEEE standard for operating systems closely related to the Unix system. A standards body that sets standards of APIs that is approved by X/Open to ensure that a POSIX approved code will run consistently on any POSIX compile platform.

portable settings Customized settings for a user's workplace that can be "transferred to" one or more additional workstations.

POSIX Portable Operating System Interfaces for Computer Environments.

Power Management A software subsystem that extends battery life in portable computer systems and conserves electrical energy (in compliance with the EPA's Energy Star guidelines) for all non-battery powered systems.

presentation device drivers Device drivers that process the high-level function calls to the Presentation Manager interface and communicate with the physical devices or the display hardware.

Presentation Manager (PM) The Workplace services that present a graphics-based interface to applications.

Presentation Manager Automated Test Environment An automated test tool for PM applications. This tool is part of the toolkit and records user activity such as moving the mouse and opening and closing windows that can be played back later.

presentation object A SOM-based object that corresponds to PM window controls such as buttons, menus (pull-downs and pop-ups), frames, entry fields, lists, title bars, scroll bars, forms, scales, and selections.

Print/Spooler Services A Workplace shared service that provides print spooling functions.

private object Object represented in a fashion that is specific to a service's implementation of the Object Manager and is unspecified to the client. Such objects are accessed only indirectly (by means of interface functions).

privilege level A method of protection that allows only certain program instructions to be used by certain programs.

procedural Pertaining to a programming system design that allows interactions between subsystems through direct calls. Contrast with object-oriented designs that require messaging facilities for such interactions.

program-file object An object that starts a program. Program files commonly have extensions of .EXE, .COM, .CMD, or .BAT.

program group Several programs that can be acted upon as a single entity.

program object An object representing the file that starts a program. You can change the settings for this object to specify the gesture assignments for the particular program.

puck A device used to select a particular location on a tablet.

Glossary - Q

query (1) A request for data from a database, based on specified conditions; for example, a request for availability of a seat on a flight reservation system. (T) (2) The process by which a master station asks a slave station to identify itself and to give its status. (T)(3) A request for information from a file based on specific conditions; for example, a request for a list of all customers whose balance is greater than \$1000.

Glossary - R

RDN Relative distinguished name.

registry A system facility for the use of each framework and its service providers. It provides a programming interface, allowing its clients to access status and control over each framework and its services. It is implemented as a general purpose data store.

remote initial program load (RIPL) (1) The initial program load of a remote requester by a server on which the appropriate information is located. (2) Supplies a remote requester with the data and programs required to boot the remote requester with an operating system. The data and programs are located on and supplied by a server. The operating system can be DOS or OS/2.

resolution In computer graphics, a measure of the sharpness of an image, expressed as the number of lines and columns on the display screen or the number of pels per unit of area.

resource script file One or more resource statements that define the type, identifier, and data for each resource.

resource servers Servers that manage and provide access to operating system entities (such as files, lists of user names, named pipes, and semaphores). Some resource servers are also name servers for the entities they control (Such as File Server).

return code (1) A code used to influence the execution of succeeding instructions. (A) (2) A value returned to a program to indicate the results of an operation requested by that program.

REXX/2 Restructured extended executor programming language for OS/2. A general-purpose programming language for command procedures, application front-ends, and user-defined macros, or subcommands. REXX programs are executed by a language processor (interpreter); that is, the program is executed line-by-line and word-by-word, without first being translated to another form (compiled).

ring level See *privilege level*.

RIPL Remote Initial Program Load.

RISC Reduced instruction set computer.

RM client The portion of the Registry Manager that executes in the task space of the client. There may be one or more RM clients in the system. Each RM client communicates with the RM server to execute tasks on behalf of the client.

root name server A Microkernel service that controls the first level of the name space. It does not support persistent information, and implements a subset of the Workplace Name Services API.

run-time Synonym for execution time. (T) (A)

Glossary - S

scatter/gather I/O operations between devices and data in non contiguous memory locations. Scatter: reads data from a device and writes to non contiguous memory locations. Gather: reads data from non contiguous memory locations and writes to a device.

screen The physical surface of a display device upon which information is shown to a user.

scroll To move a display image vertically or horizontally to view data that otherwise cannot be observed within the boundaries of the display

screen

SDE Software Development Environment.

security attribute An information item, optionally stored with a registry entry (typically by the owner of the entry), that is used by the Registry Manager in making access decisions.

security policy facility (SPF) The security object that makes security access decisions. The interface to this facility is a series of APIs defined in the security documents. The SPF centralizes security access decisions for ease of security evaluation and auditing.

security token Information string inserted into the message header of each message that is passed between tasks.

server (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (T) (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, a mail server. (A) (3) In the AIX operating system, an application program that usually runs in the background and is controlled by the system program controller. (4) In AIX Enhanced X-Windows, a program that provides the basic windowing mechanism. It handles interprocess communication (IPC) connections from clients, demultiplexes graphics requests onto screens, and multiplexes input back to clients. (5) See name server. (6) In TCP/IP, a system in a network that handles the requests of a system at another site, called a client-server.

service A managed set of useful functions intended for use by multiple clients. The Registry Manager is an example of a service.

service manager Installation programs, remote administrators, or other types of clients that can interact with (get and put) the state of frameworks and service providers through the Workplace Name Services.

service provider See *framework service provider*.

service provider interface (SPI) An interface defined by a framework for service providers. See also *framework service provider*.

settings Characteristics of objects that the user can view and sometimes alter.

setup object An object that provides a means of communication between a user and a piece of equipment or a system.

SGML Standard Generalized Markup Language.

shutdown The procedure required before turning off the computer to ensure that data and configuration information are not lost.

simple expression A sequence of parentheses, numbers, macros, and unary and binary operators that can be evaluated algebraically into an integer.

Single Byte Character Set (SBCS) A character set whose character codes are represented by one-byte code. In contrast to Double Byte Character Set.

single-user Pertaining to use by a single user (for example assumes the user is always the same; it does not distinguish between different users at different times).

slider A control that represents a quantity and its relationship to the range of possible values for that quantity. For example, a slider might indicate a time range from 0 to 10 minutes. See also *slider arm*.

slider arm The visual indicator in a slider that shows that the numerical value can be changed by manipulating it. See *slider*.

Smart Fit partition definition A function of the installation object that knows sizes and names of all the pieces that are going to be installed, including temporary space requirements for processing lists as well as size requirements for service catalogs and registries.

SMP Symmetric multiprocessing.

Socket Services A BIOS-level software interface in the PCMCIA architecture that provides a way to access the PCMCIA sockets (slots) of a computer. Socket Services identifies how many sockets are in a computer system and detects the insertion and removal of a PC Card adapter while the system is powered on.

SoftGL An open 3-dimensional imaging model standard.

Software Development Environment (SDE) A framework used to integrate the use of tools in a customizable manner on the OS/2 desktop.

SOM System Object Model.

SOM/2 Implementation of SOM for OS/2.

source directory The directory from which information is read. Contrast with *target directory*.

SPF Security policy facility.

SPI Service provider interface.

spreadsheet A worksheet arranged in rows and columns, in which a change in the contents of one cell can cause electronic recomputation of one or more cells, based on user defined relations among the cells. (A)

Standard Generalized Markup Language (SGML) An ISO standard created by IBM. A generic encoding of information; a metalanguage that provides a grammar or set of rules by which a document interchange technology can be constructed. DTDs define a common set of tags and the relationships that may exist among those tags in a valid instance of that document type.

stanza file A type of control file that has an internal script that defines devices and the order of their processing. A stanza consists of a paragraph descriptor, paragraph attributes, and a terminator.

string (1) A sequence of elements of the same nature, such as characters considered as a whole. (T) (2) In programming languages, the form of data used for storing and manipulating text.

subclass A class that is derived from another class. The subclass inherits the data and methods of the parent class and can define new methods or override existing methods to define new behavior not inherited from the parent class. See also *inheritance*.

submenu A menu related to and reached from a main menu.

symmetric multiprocessing (SMP) Equal distribution of tasks across multiple processors.

System Object Model (SOM) A mechanism for language-neutral, object-oriented programming in the OS/2 environment.

Glossary - T

tablet (1) A special flat surface with a mechanism for indicating positions thereon, normally used as a locator. (I). (2) See also *puck*.

tap To briefly touch a touch-sensitive surface with a pointing device and then quickly remove it.

target The location to which the information is destined.

target directory The directory to which information is written. Contrast with *source directory*.

TCB Trusted computing base.

TCP/IP Transmission Control Protocol/Internet Protocol

template A pattern to help the user identify the location of keys on a keyboard, functions assigned to keys on a keyboard, or switches and lights on a control panel.

Test Server A replacement for the existing "hooking" mechanism for record and playback. A message-passing architecture between the event/window services and another trusted server.

toggle To switch between two modes; for example, to switch between selected and deselected mode.

tool bar A bar containing choices that represent tools. When you select a choice from the bar, the defined action for the choice occurs.

touch area Synonym for hot spot.

touch-down point Location plotted by a digitizer where contact is made with a touch-sensitive device.

touch-sensitive Pertaining to a device such as a keypad or screen that generates coordinate data when a pointing device approaches or contacts the surface, thereby allowing a user to interact directly with a computer without entering commands from a keyboard.

track ball Synonym for control ball.

Transmission Control Protocol/Internet Protocol (TCP/IP) A set of communication protocols that supports peer-to-peer connectivity functions for both local and wide-area networks.

trigger An attribute value in a private object that tells the Registry Manager which interface (such as a shared library entry-point name) to call to get the value to be returned to the client in a public object. Used by the registry to get current state information dynamically from the service provider objects, rather than storing this data in the registry itself. When a framework is activated by a client, it searches the registry locating all instances of service providers for the particular framework. The framework then verifies the state of the service providers, one at a time, and, if required, requests that the state be brought *online*.

trusted computing base (TCB) In computer security, all of the protection mechanisms within a computer system, including hardware, software, and firmware, the combination of which enforces a security policy. It creates a basic protection environment and provides additional user services required for a trusted computer system.

Glossary - U

UA User attribute

UI User interface

unicode Code that is independent of language and culture, and supports multiple simultaneous character sets.

unit number A number used by a device driver to select a specific logical device. A unit number is unique only within the scope of the driver.

Universal Language Support See *national language support*.

UNIX ** operating system An operating system developed by Bell Laboratories that features multiprogramming in a multi-user environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers.

Note: The AIX operating system is IBM's implementation of the UNIX operating system.

user attribute (UA) A collection of <name, type, value> types. OS/2 calls its user attributes *extended attributes* or EAs.

User Defined Character (UDC) All of double byte character set and unicode define a range of character codes where a user can create and define their own characters.

User ID User identification.

User identification A short name (or identifier) that is uniquely associated with a Voice Type Dictation user.

Glossary - V

VGA Video graphics array

virtual file system A remote file system that has been mounted so that it is accessible to the local user.

vital product data (VPD) Information that uniquely defines system, hardware, software, and microcode elements of a processing system.

vnode Virtual i-node. An object in a file system that represents a file. Unlike an i-node, there is no one-to-one correspondence between a vnode and the file system; multiple vnodes can refer to a single file. A vnode is used to communicate between the upper half of the file system and the file system representations.

VPD Vital product data.

Glossary - W

wide area network (WAN) A data communications network designed to serve an area of hundreds or thousands of miles; such as public and private packet-switching networks and national telephone networks. Contrast with *local area network* (LAN).

window (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area of the screen with visible boundaries within which information is displayed. A window can be smaller than, or the same size as, the screen. (3) A division of a screen in which one of several programs being executed concurrently can display information.

window class The grouping of windows whose processing needs conform to the services provided by one window procedure.

window class style The set of properties that apply to every window in a window class.

Window's Open System Architecture (OSA) A procedural-based framework, defined by the Microsoft Corporation.

window template A data structure used to define a window.

WIN-OS/2* A feature of OS/2 that enables OS/2 to run supported Windows** programs.

Workplace An architecture for a family of products, built on the IBM Microkernel, that support multiple coexistent personalities.

Workplace Shell* interface The object-oriented, graphical user interface of OS/2 2.0 and later versions of the operating system.

Glossary - X

XMS Extended memory specification.

XOM OSI Object Management developed by X/Open. Part of X/Open's Common Application Environment (CAE) specification. XOM is also known as *OSI-Abstract-Data-Manipulation API* . It is a generalized framework for abstracting the physical representation of different classes of objects. XOM class libraries are called *packages* . Used extensively in the implementation of the Workplace Name Services.

X/Open Portability Guide Issue 4 A standards group developed by X/Open that sets standards for APIs to ensure consistency.

XPG4 X/Open Portability Guide Issue 4.

XPTB A framework architecture in which frameworks and service providers use a registry to install and configure themselves.

XTERM window In X-windows, a console window that displays Stdout/Stderr output.

Glossary - Y

There are no glossary terms for this starting letter.

Glossary - Z

There are no glossary terms for this starting letter.
